313-CD-610-003

EOSDIS Core System Project

Release 6B ECS Internal Interface Control Document for the ECS Project

March 2003

Raytheon Company
Upper Marlboro, Maryland

This page intentionally left blank.

Release 6B ECS Internal Interface Control Document for the ECS Project

March 2003

Prepared Under Contract NAS5-60000
CDRL Item #051

RESPONSIBLE ENGINEER

Alton Davis /s/	3/19/03
Alton Davis, Systems Engineer EOSDIS Core System Project	Date

SUBMITTED BY

Arthur Cohen /s/	3/19/03
Arthur Cohen, Director of Development EOSDIS Core System Project	Date

Raytheon Company
Upper Marlboro, Maryland

313-CD-610-003

This page intentionally left blank.

Preface

This document is the final version of a formal contract deliverable with an approval code 3. As such it is reviewed and controlled by the contractor. Contractor approved changes to this document are handled in accordance with change control requirements described in the ECS Configuration Management Plan. Changes to this document are made by Document Change Notice (DCN) or by complete revision. This document has been reviewed by the ECS Development Facility/Science Development configuration control board and supports the "as built" Release 6B System. Any questions or proposed changes should be addressed to:

Data Management Office
The ECS Project Office
Raytheon Systems Company
1616 McCormick Drive
Upper Marlboro, MD 20774-5301

This page intentionally left blank.

Abstract

This document provides a set of interface scenarios that describe how the Release 6B ECS interacts to execute end-to-end system threads. A domain (or end user) view and a component interaction view are presented for each scenario. This document is intended for application software engineers, systems engineers and system maintenance engineers to understand how CSMS/SDPS components interact to perform key system functions. For detailed internal interface information, on-line output provided by automatic software tools such as Discover and ABC++ should be used.

The scenarios in this document reflect the capabilities and functions of the "as built" design for Drop 6B.

Keywords**:** external interface, internal interface, public class, private class, class category, system-level scenario, scenario primitive, interface class, CCS Middleware

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Submitted as Final
iii through xx	Submitted as Final
1-1 and 1-2	Submitted as Final
2-1 and 2-2	Submitted as Final
3-1 through 3-512	Submitted as Final
AB-1 through AB-16	Submitted as Final

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
313-CD-610-001	Submitted as Draft	August 2000	
313-CD-610-002	Submitted as Final	August 2002	02-0668
313-CD-610-003	Submitted as Final	March 2003	03-0114

This page intentionally left blank.

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

1. Introduction

1.1 Identification

This Release 6B ECS Internal Interface Control Document (ICD) for the ECS Project, Contract Data Requirement List (CDRL) item 051, with requirements specified in Data Item Description (DID) 313/DV3, is a required deliverable in its final form under the Earth Observing System (EOS) Data and Information System (EOSDIS) Core System (ECS), Contract (NAS5-60000), Revision C, October, 1999 (423-41-03).

1.2 Scope

The Release 6B Internal ICD specifies software interfaces internal to the CSMS/SDPS software architecture. It defines Release 6B services in the context of system level scenarios. The relationships and interactions between the Release 6B CSCIs are presented. This document also describes how ECS infrastructure services are used by the ECS internal applications.

This document addresses all interface classes from SDPS and CSMS CSCIs, which are linked to create a desired scenario. External interfaces are mapped to the internal ECS object(s) that provide the service.

This document is not intended to include hardware interface information between subsystems or hardware descriptions. Subsystem hardware is described in the DID 305 document identified in sub-section 2.2. Any reference to hardware processes in this document is meant to portray functional activity relative to software processes and not specific hardware functions.

This document describes the ECS system in terms of its support of several primary scenarios. These scenarios, based on the normal support of EOS instruments, are listed below and are described in Section 3:

- ESDTs (Earth Science Data Types)
- System Startup/Shutdown
- MODIS (an instrument on the Terra spacecraft which provides data to three DAACs)
- Landsat-7
- ASTER (an instrument on the Terra spacecraft which provides data to Japan (GDS)).
- Planning Scenarios
- EDOS/ FDD/EMOS Interfaces
- Cross Mode/DAAC Scenario
- Science Investigator-Led Processing Systems (SIPS) Scenario
- Fault Recovery

1.3 Document Organization

The document is organized to describe the Release 6B internal interfaces.

Section 1 provides information regarding the identification, scope, status, and organization of this document.

Section 2 provides a listing of the related documents, which were used as source information for this document.

Section 3 contains the system level scenarios illustrating the interactions between the ECS CSCIs/CSCs. This section also provides an overview of the interface modeling approach to document the internal interfaces.

2. Related Documentation

2.1 Parent Documents

194-207-SEI	System Design Specification for the ECS Project
313/DV3	ECS Internal Interface Control Documents

2.2 Applicable Documents

305-CD-610	Release 6B Segment/Design Specification for the ECS Project
311-CD-606	Subscription Server Database Design and Schema Specifications for the ECS Project
611-CD-610	Mission Operation Procedures – Drop 6B
625-CD-604	ECS Project Training Manual, Volume 4: System Administration
505-41-32	ESDIS Document, Interface Control Document between the EOSDIS Core System (ECS) and the Landsat-7 System

2.3 Information Documents Not Referenced

The documents listed below, while not directly applicable, do help in the maintenance of the delivered software.

423-41-02	Goddard Space Flight Center, Functional and Performance Requirements Specification for the Earth Observing System Data and Information System Core System
423-41-57	ICD between ECS and the Science Investigator - Lead Processing Systems (SIPS)
423-41-63	ICD between EMOS and SDPS
505-41-40	ICD between ECS and GSFC DAAC
540-022	Goddard Space Flight Center, Earth Observing System Communications System Design Specification Interface Requirements Document
560-EDOS-0211.0001	Goddard Space Flight Center, Interface Requirements Document Between EDOS and the EOS Ground System Operational Agreement between the Landsat 7 Data Handling Facility and the Distributed Active Archive Center (DAAC) at the EROS Data Center (EDC)

This page intentionally left blank.

3. Interface Scenarios

3.1 Overview

The purpose of this section is to document how ECS works to fulfill its mission. The ECS mission is, in its essence, to manage Earth Science-related data in the following ways:

- To receive data from external sources,
- To save that data in either long-term or permanent storage,
- To produce higher level data products from the received data, and
- To support access to the data by Earth Scientists as well as other registered clients

ECS is a complex software system that comprises nine subsystems. Each subsystem comprises a set of software programs (COTS and custom built) working together to exchange information and control the management of Earth Science-related data.

A preferred method to document how a complex system such as ECS works is to follow a specific thread of functionality, or scenario, tracing how the ECS Clients (both human and software) and internal ECS ***components*** interact in support of the scenario. The interaction between the ECS components can be understood by focusing on how the interfaces offered by the ECS components are used in support of the system functionality required to support the given scenario.

This section documents one facet of a multi-faceted problem. To get a more complete view of precisely how each ECS component performs its role, the reader should also reference the design material presented by each of the ECS components. This material can be found in DID 305[1]. Table 3.1-1 maps the subsystems and their components to their appropriate interface process. Only major interface processes utilized in the scenarios are shown in this table. Indeed, this document and CDRL 305 should be used in conjunction with each other. CDRL 305 provides a general description of the processes and a statement of what the components are providing and how they provide it. This section documents how those components work together in specific applications to provide a complete system.

It should be noted that many of the scenarios involve a software component/operations interface with a human operator. The intent of the descriptions of these interfaces is to show the involvement of a human operator to the extent necessary to affect the correct component function and not to show operator procedures. These procedures are detailed in the 611 document specified in sub-section 2.2.

[1] The DID 305 refers to ECS Document: 305-CD-610-003, Release 6B Segment/Design Specification for the ECS Project.

Table 3.1-1. ECS Subsystem and Component Design Overviews (1 of 3)

Subsystem (CI)	CSCI/Component	Major Interface Process
CLS	EOS Data Gateway (EDG)	Netscape
	User Profile Gateway	EcClDtUserProfileGateway
	Java DAR Tool	EcClWbJdt
	EOSView	EOSView
	Java Earth Science Server process	JESS
	User Log in	EcClOdUserLogin
	On-Demand Form Request Manager	EcClOdProductRequest
CSS	Subscription Server	EcSbSubServer
	Subscription Server GUI	EcSbGui
	Ftp Server	ftp_popen
	Configuration Registry Server (CRS)	EcCsRegistry
	ASTER DAR Communications Gateway	EcGwDARServer
	ASTER E-mail Parser Gateway	EcCsEmailParser
	MOJO Gateway	EcCsMojoGateway
	Machine to Machine Gateway	EcCsMtMGateway
	Sybase Adaptive Server Enterprise (ASE - COTS for Subscription Server)	N/A
DMS	V0 Gateway	EcDmV0ToEcsGateway
	ASTER Gateway	EcDmEcsToAsterGateway
		EcDmAsterToEcsGateway
	Maintenance Tool - Mtool	EcDmDdMaintenanceTool
	Data Dictionary (DDICT)	EcDmDictServer
	Sybase ASE (COTS for DDICT)	N/A
	PDSIS Server	pdsis.jar – start script
		EcPdPDSISServerStart
DPS	AutoSys (COTS)	Event_daemon
	Job Management	EcDpPrJobMgmt, EcDpPrJobMgmtClient
	DPR Execution	EcDpPrEM, EcDpPrRunPGE
	DPREP	EcDpPrAm1FddAttitudeDPREP_PGE
		EcDpPrAm1EdosAncilliary
		EcDpPrAm1EdosEphAHDPREP_PGE
		EcDpPrAm1EdosEphemerisRepair
		EcDpPrPM1AttitudeDPREP_PGE
		EcDpPrPm1FddEpehemerisDPREP_PGE
		EcDpPrAm1ToolkitToHdf
		EcDpPrDumpAttitudeDPREP
		EcDpPrDumpEphemerisDPREP
		EcDpPrEMGetAncHeaders
	Ground Event	EcDpPrGE
	QA Monitor	EcDpPrQaMonitorGUI
	AITTL	EcDpAtMgr, EcDpAtSSAPGui, EcDpAtInsertExeTarFile
	Deletion	EcDpPrDeletion

Table 3.1-1. ECS Subsystem and Component Design Overviews (2 of 3)

Subsystem (CI)	CSCI/Component	Major Interface Process
DSS (SDSRV)	Science Data Server	EcDsScienceDataServer
	HDF EOS Server	EcDsHdfEosServer
	Science Data Server Operator GUI	EcDsSdSrvGui
	Granule Deletion Administration Tool	EcDsGranuleDelete
	Sybase ASE/SQS (COTS for SDSRV)	N/A
	PDSIS	pdsis.jar – PDSIS SCLI cron job
DSS (DDIST)	Data Distribution Server	EcDsDistributionServer
	Data Distribution Operator GUI	EcDsDdistGui
	Sybase ASE (COTS for DDIST)	N/A
	PDSIS	pdsis.jar – PDSIS cron job polls DDIST Database
DSS (STMGT)	Archive	EcDsStArchiveServer
	Request Manager	EcDsStRequestManagerServer
	Staging Disk	EcDsStStagingDiskServer
	Cache Manager	EcDsStCacheManagerServer
	Storage Management Operator GUI	EcDsStmgtGui
	Ftp Server	EcDsStFtpServer
	DTF Tape	EcDsStDTFServer
	Pull Monitor	EcDsStPullMonitorServer
	Sybase ASE (COTS for STMGT)	N/A
	AMASS (COTS for STMGT)	N/A
INS	Polling Ingest	EcInPolling
	Ingest GUI	EcInGUI
	Request Manager	EcInReqMgr
	Granule Server	EcInGran
	Ingest E-mail Gateway Server	EcInEmailGWServer
	Sybase ASE (COTS for configuration and state)	N/A
MSS	User Registration Server	EcMsAcRegUserSrvr
	User Registration Server (SMC) GUI	EcMsAcSMCRegUserGUI
	User Registration Server (Home DAAC) GUI	EcMsAcDAACRegUserGUI
	Order Tracking Server	EcMsAcOrderSrvr
	Order Tracking GUI	EcMsAcOrderGUI
	Use Profile Database (Sybase ASE - COTS)	N/A

Table 3.1-1. ECS Subsystem and Component Design Overviews (3 of 3)

Subsystem (CI)	CSCI/Component	Major Interface Process
PLS	Subscription Editor	EcPlSubsEdit
	Production Request Editor	EcPlPREditor_IF
	Subscription Manager	EcPlSubMgr
	Production Planning Workbench	EcPlWb
	Resource Planning Workbench Editor	EcPlRpRe
	Resource Planning Workbench Reservation Editor	EcPlRpSi
	Resource Planning Workbench Timeline	EcPlRpTl
	Workbench Timeline Tool	EcPlTl
	Sybase ASE (COTS for PDPS database)	N/A
	On-Demand Production Request Mgr.	EcPlOdMgr
PDS	V0 GTWAY	EcDmV0ToEcsGateway
	SDSRV	EcDsSCLI
	DDIST	EcDsDistributionServer
	Oracle Server	N/A
OM	V0 GTWAY	EcDmV0ToEcsGateway
	SDSRV	EcDsScienceDataServer
	PDSIS	pdsis.jar – start script EcPdPDSISServerStart
	Sybase ASE	N/A

3.2 Scenario Approach

Section 3.3 describes the steps required prior to the start of usage of the EOSDIS system. The steps taken to install, update, import and export ESDTs are defined in this section.

Section 3.4 provides the document names, numbers and sections where the startup and shutdown procedures and process are documented. [This scenario is now described in the Mission Operations Procedures document, 611-CD-600, Section 3.2. Also, in the ECS Project Training Material document, Volume 4: System Administration, 625-CD-504, in the section titled System Startup and Shutdown, the process for start up and shutdown is described.]

Sections 3.5 - 3.7 document the ECS system in terms of its support of three primary scenarios using the following primary EOS instruments. (Note other instruments use the same interfaces and are not shown).

- MODIS
- Landsat-7
- ASTER

Section 3.8 describes the Production Planning scenario. This scenario applies to processing common to MODIS and ASTER scenarios.

Section 3.9 describes the EDOS/FDD/EMOS Interfaces Scenario.

Section 3.10 describes threads utilized for cross Mode and cross DAAC data transfers.

Section 3.11 describes the support for SIPS

Section 3.12 outlines client and server failure recovery policies and processes.

Sub-sections describe how ECS supports each of these scenarios in the above sections from two perspectives, the domain view and the component view. The domain view breaks the scenario into a sequence of activities based upon what happens from the Operational or Science Data perspective. This view presents how ECS-external users and systems interact with ECS as well as looking at how the science data is managed within ECS. This view does not present the details of specific process interactions. The component view shows a more detailed set of interactions that describe the interface usage between ECS components. Each interaction between components is documented, in terms of how and why. Each of the scenarios documented here has been partitioned into primary threads of activity. Each thread of the scenario is documented independently to simplify the scenarios.

3.2.1 Scenario Presentation Approach

This section describes how the ECS support of each scenario is presented. As mentioned above, each Scenario is partitioned into a sequence of threads of activity. Each of those threads is documented in the same manner. The following paragraphs define this documentation approach.

Scenario Description: First, each scenario is described from the science mission perspective. The primary system functions being exercised are identified.

Scenario Preconditions: All activities that must have been performed prior to the execution of the scenario are identified.

Scenario Partitions: The scenario threads are identified and described.

Scenario Thread Interaction Diagram: A diagram is presented for each Scenario Thread. This diagram shows external system, ECS User, DAAC Operator and ECS-internal subsystem interactions. The notation of the diagram allows for the interactions to be labeled with numbers and short terms (events). The arrow numbering uses the convention of a letter, representing the Thread within the Scenario, and a sequence number (e.g., A.1, A.2, B.2, B.3). The mechanism of the interactions (e.g. CCS Middleware, HMI, ftp, and e-mail or as noted) is identified by the interaction line presentation style. The arrow direction indicates process flow direction for the event named. Note that *request* events show the direction of the request and the return answering the request is assumed and not necessarily shown as a separate interface direction.

Interaction Table - Domain View: Each Scenario Thread is documented in a table, which describes the interactions presented in the Scenario Thread Interaction Diagram. These interactions are not the detailed definitions of how the interactions are fulfilled, but rather that they need to occur. This table further specifies the interactions as each row represents an

interaction event. The columns in the table delimit how each interaction is defined. The Interaction Table - Domain View includes the following columns:

Step: An identifier of the step within the Scenario Thread. Each step is identified with an "x.y" label, where x is a letter referring to the Thread within the scenario, and y is a sequence number.

Event: The name of an interaction occurrence between major parts of the system (i.e., Subsystem to Subsystem).

Interface Client: The Client of the interaction. This can be viewed as who is asking the question, or who is stimulating the action. Included in this column are Users, Operators, External Systems and usually ECS subsystems rather than components.

Interface Provider: All Interactions are described in terms of exercising well-defined interfaces. Those interfaces are offered by some entity in the system and are similar to those identified as Interface Clients. The Interface Provider is not only responsible for offering the interface, but for ensuring that the interface is met. The provider is doing the action required, perhaps collaborating with other system entities.

Data Issues: This column describes any special Data related issues. This description includes the data types, volumes and frequencies, as well as the current source of the data used in the system. The word "None" indicates there are no data issues.

Step Preconditions: Any special preconditions that must have been met for the interaction to be successful are called out here. The word "None" indicates there are no special preconditions for this particular step. It is assumed that all baseline subsystems (DSS, INS, CLS, DMS, PLS, DPS, CSS and MSS) are up and running for each thread.

Description: A description is given of what generally occurs during the interaction, as well as its application in this scenario step.

Component Interaction Table: Each Scenario Thread is further documented in the Component Interaction Table. This table specifies each ECS component-level interaction that is required to support the steps in the Scenario Thread.

Each of these interactions is numbered in a way consistent with the Scenario Thread it is supporting. Specifically, each Component Interaction step is numbered with a "sub"step number in a sequence within that Scenario Thread step. For example, if there are three component interactions required fulfilling Scenario Thread step A.3, those three steps are numbered A.3.1, A.3.2 and A.3.3. Please note that if no component interaction is required to fulfill a Scenario Thread Step (i.e. - only human-to-human interaction), there are no component interaction steps. Therefore, in the Component Interaction steps, a Scenario Thread Step might be skipped.

Each row in the Component Interaction Table defines a step in how the system supports the capability. The columns in the Component Interaction Table are:

Step: An identifier, as described above, of the step within the Scenario Thread.

Event: The name of an interaction occurrence between components.

Interface Client: The Client of the interaction. This can be viewed as who is asking the question, or who is stimulating the action. Included in this column are Users, Operators,

External Systems and ECS components. Where ECS components are the Interface Clients, the specific component process is identified.

Interface Provider: This identifies the entity in the system providing the interface used to perform some capability. Interface Providers are primarily ECS Components, which are identified by the component process name.

Interface Mechanism: This column identifies how the interface is accomplished. It defines the low level (normally software) mechanism used by the Interface Client and Provider to exchange necessary information. This is also shown in the scenario diagrams for only the particular component interaction between subsystems – consult the key.

Description: This column contains text describing what is occurring during this step. It describes what is occurring in the context of this scenario thread. It describes not only what is happening, but also how it happens and how the client knows how to ask for it.

3.2.2 Scenario Process Flow

The ECS Science Data System is a complex collection of subsystems. There is no single path through the many features of the system. However, there is a general logical flow through the various capabilities. Figure 3.2.2-1 describes the key elements of this flow. Each of the elements identified is described in more detail in the individual scenario threads.

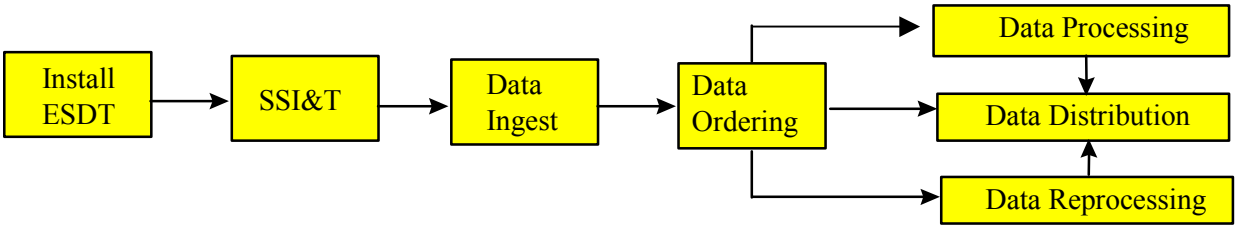

Figure 3.2.2-1. Scenario Process Flow

Install ESDT

All data interactions within the ECS are performed against Earth Science Data Types (ESDTs). An ESDT is the logical object describing both the inventory holdings for particular data, and the services (insert, acquire etc.) that can be applied to the data. Before a user (including DAAC operations) can perform any data services against a data set in the ECS, the ESDT for the data type must be installed. Installation includes defining the collection level and granule level metadata in the inventory (Science Data Server), advertising the data type and it's services, defining metadata attribute valids in the Data Dictionary, and defining data events for subscription.

SSI&T

Science Software Integration & Test (SSI&T) is the process Instrument Team developed algorithms get qualified for use in the ECS production environment. Much of this process is algorithm specific and/or manual or semi-automatic. These aspects are not dealt with in this document. However, the reception of the original algorithm package (Delivered Algorithm

Package - DAP) and the qualified algorithm package (Science Software Archive Package - SSAP) are automated tasks, and are covered in detail in the scenarios.

Data Ingest

Once the ESDT is defined, and any production algorithms have been integrated, the ECS is ready to accept data and generate higher-level products. This document covers a number of different data ingest scenarios that occur with ECS data.

Data Ordering

There are a number of ways data can be requested in the ECS. If the product exists in the archive, a user can simply request it for distribution. If the product doesn't exist, but the algorithms for producing the product have been integrated, the user can request production. Alternatively, if the product exists, but has been generated with a set of run-time parameters different from those desired, the user can request the product be reprocessed.

Data Processing

Many products are produced automatically upon the availability of the necessary input data. But in addition to this 'standard' production, the ECS also has the capability to produce products on-demand in response to a user request. Both types of production, together with QA processing and algorithm failure handling are described in detail in the scenarios.

Data Reprocessing

An important feature of the ECS is the capability to reprocess data when either the original production data or algorithm was faulty, or if another user needs different execution parameters.

Data Distribution

Once data is generated or ingested into the ECS, it is made available to other users. The distribution system provides for a flexible data delivery system providing data either automatically based on pre-established event triggers or by direct request.

3.2.3 Error Handling and Processing

EcUtStatus is a class used throughout the ECS custom code for general error reporting. It is almost always used as a return value for functions and allows detailed error codes to be passed back up function stacks.

The following CSCIs perform error handling and processing via this class and other classes, functions, and utility programs:

Data Server Subsystem (Science Data Server CSCI)

DsShError is a Science Data Server specific class used mainly for exception handling.

DsShErrorDetails is a Science Data Server class that can be used to convert error details (in an EcUtStatus object) into more meaningful text messages.

The Science Data Server uses two main mechanisms for error handling.

1. Return Values

Functions can return an EcUtStatus object, which can be used to indicate a general success/failure status. Also, more detailed information on the exact reason for the failure can be provided. For example, a granule cannot be acquired because it has restricted access privileges. This is the most widely used mechanism within the Science Data Server and in general these errors get propagated back up to the top-level functions with ALOG error messages being generated along the way.

2. Exceptions

Some functions (for example, class constructors) cannot return values to indicate success or failure. These functions may throw exceptions, usually instances of the DsShError class. These errors are usually caught by other functions at a low level and converted into EcUtStatus return values (as described in 1).

In addition, the DsShErrorDetails class can be used to map error values (as contained in an EcUtStatus object) into text messages. This enables better reporting of errors in the Science Data Server logs.

Currently, the Science Data Server client interface only supports returning error messages back to client programs, along with a generic success/failed status.

For writing messages to the Applications Log (ALOG), the following functions are used:

DsLgLogError sends a message to the ALOG at severity level 1. For example, DsLgLogError ("DsMdMetadataCheckpoint1", "Bad granule UR");

DsLgLogWarning sends a message to the ALOG at severity level 2. For example, DsLgLogWarning ("DsMdMetadataCheckpoint2", "Unable to retrieve granule metadata");

DsLgLogInformational sends a message to the ALOG at severity level 3. For example, DsLgLogInformational ("DsMdMetadataCheckpoint3", "Failed to construct granule");

For writing messages to the debug log, the following macros are used:

PF_STATUS writes a message at a "log level" of 1 to the debug log. For example, PF_STATUS {cerr << "Issue rpc to STMGT" << endl;}

PF_VERBOSE writes a message at a "log level" of 2 to the debug log. For example, PF_VERBOSE {cerr << "Request received from client" << endl;}

PF_DEBUG writes a message at a "log level" of 3 to the debug log. For example, PF_DEBUG {cerr << "Saved request to database" << endl;}

Data Server Subsystem (STMGT CSCI)

The Storage Management (STMGT) CSCI uses the following classes to deal with errors:

The class EcUtStatus is used to hold the actual error number when an error occurs. The EcUtStatus object is returned to the STMGT clients when the request is complete. The class DsStErrorDetails is used to extract the error type and severity information from the EcUtStatus class.

The class DsShLogging is used throughout the STMGT code to control the writing of error messages to the application log files and debug log files. This class uses the ECS standard logging functions and macros. Errors can also be logged to the DsStEventLog table within the STMGT database. These errors can be viewed with the STMGT GUI. These errors are periodically removed by the DsStRequestManager Server / GRCleanup Stored Procedure.

Here is an example of how logging is used:

```
EcUtStatus status;

.

.

.

if (somethingWentWrong)

{

ourDsStLogging.LogAndSetError(status, DsEStInternalSybError,

"DBIF:FetchQueryResult: ct_fetch failed");}
```

The program EcDsStErrorFilesGenerator is used to generate the various error codes used by STMGT into the STMGT database.

Data Server Subsystem (DDIST CSCI)

The class EcUtStatus is used to hold the actual error number. The EcUtStatus object is returned to the DDIST clients when the request is complete.

The Data Dictionary Server CSCI uses two main mechanisms for error handling.

1. Return Values

Functions can return an EcUtStatus object, which can be used to indicate a general success/failure status. Also, more detailed information on the exact reason for the failure can be provided. This is the most widely used mechanism within the Data Dictionary Server and in general these errors get propagated back up to the top-level functions with ALOG error messages being generated along the way.

2. Exceptions

Some functions (for example, class constructors) cannot return values to indicate success or failure. These functions may throw exceptions, usually instances of the DsShError class. These errors are usually caught by other functions at a low level and converted into EcUtStatus return values (as described in 1).

In addition, the DsShErrorDetails class can be used to map error values (as contained in an EcUtStatus object) into text messages. This enables better reporting of errors in the DDIST logs.

Currently, the DDIST client interface only supports returning error messages back to client programs, along with a generic success/failed status.

For writing messages to the Applications Log (ALOG), the following functions are used:

EcLgLogError sends a message to the ALOG at severity level 1. For example,

```
catch (DsShError& err)

{EcLgLogError ("DsDdSchedulerExecuteFunctionError", 0, err.GetMsg());}
```

EcLgLogInformational sends a message to the ALOG at severity level 3. For example,

```
EcLgLogInformational ("ConfigVarMissing", status.GetLogMessageLink(),

   "EcCUtRpcClientIDConfigTag var not set in Config File.");
```

For writing messages to the debug log, the following macros are used:

PF_STATUS writes a message at a "log level" of 1 to the debug log. For example,

```
PF_STATUS{cerr << "DsDdCBCache::Create Creating first instance." << endl;}
```

PF_VERBOSE writes a message at a "log level" of 2 to the debug log. For example,

```
PF_VERBOSE{cerr << "Calling DsStArchive::Create . ReqID | FullArchiveID:\n"

        << myRequestID <<" | " <<  FullArchiveID << " and\n"

        << "RpcID = " << RpcId_CR.AsString() << endl;}
```

PF_DEBUG writes a message at a "log level" of 3 to the debug log. For example,

```
PF_DEBUG {cerr << "DsDdDistRequestS::SetPriority.  Request: "

        <<  myRequestID << " Priority: " << Priority <<  endl;}
```

Data Processing Subsystem (PRONG & AITTL CSCIs)

The Data Processing Subsystem (DPS) uses the EcUtStatus class for general error handling. It is almost always used as a return value for functions and allows detailed error codes to be passed back up function stacks.

The PRONG and AITTL CSCIs use two main mechanisms for error handling.

1. Return Values

Functions can return an EcUtStatus object, which can be used to indicate a general success/failure status. Also, more detailed information on the exact reason for the failure can be provided. This is the most widely used mechanism within the PRONG and AITTL and in general these errors get propagated back up to the top-level functions with ALOG error messages being generated along the way.

2. Exceptions

Some functions (for example, class constructors) cannot return values to indicate success or failure. These functions can throw exceptions. These errors are usually caught by other functions at a low level and converted into EcUtStatus return values (as described in 1).

Currently, the PRONG and AITTL CSCIs client interfaces only support returning error messages back to client programs, along with a generic success/failed status.

In addition, the PRONG and AITTL use the following method of tracking errors; stored procedure tracing. Some PRONG and AITTL stored procedures write error and informational messages to a table in the Planning and Data Processing Subsystems (PDPS) database named DpPrTrace. The stored procedure ProcInsertTrace is used to do this. A trigger on this table truncates the messages after 10,000 messages have been written to the table. There are no other specialized error libraries and classes used by the PRONG and AITTL.

Column	Example
Time	Apr 2 2002 14:01:27.960000
Procedure Name	ProcGetReadyDPRs
Called From	(NULL)
Dpr Id	MODPGE02#s28021500OPS
Message	99 pgeId slots available for pgeId MODPGE02#syn#001

Example Row from the DPS DpPrTrace Table

For writing messages to the Applications Log (ALOG), the following functions are used:

EcLgLogError sends a message to the ALOG at severity level 1. For example, EcLgLogError (methodName,

returnStatus.GetLogMessageLink(),

"Error: unable to copy tar file from '%s%s' to '%s' ",

dataserverpath.data(), pgeTarName.data(),

destinationPath.data());

EcLgLogWarning sends a message to the ALOG at severity level 2. For example, EcLgLogWarning(methodName, lstatus.GetLogMessageLink(), "error terminating DPR job") ;

EcLgLogInformational sends a message to the ALOG at severity level 3. For example, EcLgLogInformational(methodName,

status.GetLogMessageLink(),"not using DpPrAutoSysProfile");

For writing messages to the debug log, the following macros are used:

PF_STATUS writes a message at a "log level" of 1 to the debug log. For example,

PF_STATUS {cerr << methodName << "Can't retrieve DpPrDeleteFailedPGEJobs";

cerr << " param. -- setting to FALSE" << endl;}

PF_VERBOSE writes a message at a "log level" of 2 to the debug log. For example,

PF_VERBOSE {cerr << methodName <<"urName = " << urName << endl;}

PF_DEBUG writes a message at a "log level" of 3 to the debug log. For example,

PF_DEBUG {cerr << methodName << "can not get PF config file pointer!" << endl;}

Planning Subsystem (PLANG CSCI)

The Planning Subsystem (PLS) uses the EcUtStatus class for general error handling. It is used mostly as a return value for functions and allows detailed error codes to be passed back up function stacks.

The PLANG CSCI uses two main mechanisms for error handling.

1. Return Values

Functions can return an EcUtStatus object, which can be used to indicate a general success/failure status. Also, more detailed information on the exact reason for the failure can be provided. For example, a granule cannot be acquired because it has restricted access privileges. This is the most widely used mechanism within the PLANG and in general these errors get propagated back up to the top-level functions with ALOG error messages being generated along the way.

2. Exceptions

Some functions (for example, class constructors) cannot return values to indicate success or failure. These functions can throw exceptions. These errors are usually caught by other functions at a low level and converted into EcUtStatus return values (as described in 1).

Currently, the PLANG client interface only supports returning error messages back to client programs, along with a generic success/failed status.

In addition, the PLANG has a class called PlDetailedError.h in /ecs/formal/PDPS/PLS/PLANG/src/Corelib to exchange information between corelib classes and the Production Request Editor Graphical User Interface (GUI).

For writing messages to the Applications Log (ALOG), the following functions are used:

EcLgLogError sends a message to the ALOG at severity level 1. For example, EcLgLogError ("CANTGETEXISTINGDEMPGEIDS", status.GetLogMessageLink(),"Failed to get existing DEM pgeIds");

EcLgLogWarning sends a message to the ALOG at severity level 2. For example, (EcLgLogWarning ("CANTRETRIEVEPRIMARYINPUTS",status.GetLogMessageLink(),"Failed to retrieve primary inputs.");

EcLgLogInformational sends a message to the ALOG at severity level 3. For example, EcLgLogInformational ("SUBMITSUBSUCCESS", 0,"Able to submit subscription for Data Type %s",myDataTypeId.data());

For writing messages to the debug log, the following macros are used:

PF_STATUS writes a message at a "log level" of 1 to the debug log. For example,

```
PF_STATUS {cerr <<"PlNotification::DbConnect" << endl;
          cerr <<"Successfully connected user " << endl;}
```

PF_VERBOSE writes a message at a "log level" of 2 to the debug log. For example,

```
PF_VERBOSE {cerr << "DPR no longer present in database.  Continuing "
                 << "to process PR." << endl;}
```

PF_DEBUG writes a message at a "log level" of 3 to the debug log. For example,

```
PF_DEBUG {cerr << "SQL statement: " << sqlStatement << endl;}
```

INGEST Subsystem (INGST CSCI)

The INGEST Subsystem (INS) uses the class EcUtStatus for general error reporting. It is used mostly as a return value for functions and allows detailed error codes to be passed back up function stacks.

DsShError is a Science Data Server specific class used by INGEST for printing the line number in the code where an error occurred.

The INGST CSCI uses two main mechanisms for error handling.

1. Return Values

Functions can return an EcUtStatus object, which can be used to indicate a general success/failure status, as well as more detailed information on the exact reason for the failure. This is the most widely used mechanism within INGST and in general, these errors get propagated back up to the top-level functions with ALOG error messages being generated along the way.

2. Debug Messages

The Ingest code uses the PF_STATUS, PF_VERBOSE and PF_DEBUG macros to log messages to the debug logs. These macros are part of the EcPfGenProcess class.

For writing messages to the debug log, the following macros are used:

PF_STATUS writes a message at a "log level" of 1 to the debug log. For example,

```
PF_STATUS {cerr << "InDataPreprocessTask::Preprocess successful.  ";
   cerr << "RequestID="<< myRequestID<< " GranuleID="<<myGranuleID <<"
CollectionName="<<myDataType<< endl;}
```

PF_VERBOSE writes a message at a "log level" of 2 to the debug log. For example,

```
PF_VERBOSE {cerr << "InFDDorbitMetadata::Preprocess() " << endl;
   cerr << "Byte-ordered FDD Orbit Metadata preprocessing successful. ";
   cerr << "RequestID="<< myRequestID << ", ";
   cerr << "GranuleID="<< myGranuleID << ", ";
   cerr << "CollectionName=" << myDataType.data() << endl;
   cerr << "myInputFile: " << myInputFile.data() << endl;}
```

PF_DEBUG writes a message at a "log level" of 3 to the debug log. For example,

```
PF_DEBUG {cerr << __FILE__ << ":" << __LINE__ << endl;
   cerr << "InDataPreprocessTask::PreprocessSceneMetadata ";
   cerr << "browseFileName = " << browseFileName << endl;
   cerr << "DataType = " << myDataType ;
   cerr << "VersionID = " << myVersionID;
   cerr << "RequestID="<< myRequestID<< " GranuleID="<<myGranuleID << endl; }
```

In addition, INGST generates the following special messages: PDRD and PAN.

1. Product Delivery Record Discrepancy (PDRD)/Data Availability Acknowledgement (DAA)

When there is a problem with a Product Delivery Record (PDR) (or Data Availability Notice (DAN), INGST generates a PDRD or DAA message. The message can be sent via e-mail and/or transferred (via FTP) if so configured. The code to generate these messages is in the InMsg library.

1. Production Acceptance Notification (PAN)/Data Delivery Notice (DDN) messages

When an INGST request has completed, INGST generates a PAN or DDN message. The message can be sent via e-mail and/or transferred (via FTP) if so configured. The code to generate these messages is in the InMsg library.

Communications Subsystem (DCCI CSCI) and

System Management Subsystem (MCI and MLCI CSCIs)

When an error occurs, the error is logged into the applications log (ALOG). The Communications Subsystem (CSS) and System Management Subsystem (MSS) have two main mechanisms to handle the error:

1. Return an error status
2. Throw an exception

The CSS uses the following classes for error handling and processing:

The EcUtStatus class is used to describe the operational status of many functions. The values most often reported are "failed" and "ok." But depending upon the application, detailed values could be set and sent. Please refer to the definition of this class (located in /ecs/formal/COMMON/CSCI_Util/src/Logging/EcUtStatus.h) for all possible values.

The EcPoError class defines the basic error types and handling functions for using the EcPoConnectionsRW class (based upon RWDBTool++). The Subscription Server and MSS Order tracking Server use the EcPoConnectionsRW class.

The RWCString is used to store some status value returned by applications.

Integer is used to return some error status by applications. This is used specifically between client and server communications.

Many types of exceptions can be sent and handled by the CSS. These include exceptions sent by Commercial Off The Shelf (COTS) products (such as DCE, RWDBTools++, and RWTools++), systems and exceptions defined by individual applications.

For writing messages to the debug log, the following macros are used:

PF_STATUS writes a message at a "log level" of 1 to the debug log. For example,

PF_STATUS {cerr << "EcSbSubServer: Host name = " << hostInfo.nodename << endl;}

PF_VERBOSE writes a message at a "log level" of 2 to the debug log. For example, PF_VERBOSE {cerr << "EcSbUpdateSubRequest:Update: Succeeded." << endl;}

PF_DEBUG writes a message at a "log level" of 3 to the debug log. For example,

PF_DEBUG {cerr << "EcSbSubscription:Execute: Action = ACQUIRE : " << endl;

cerr << *tmpAction << endl;}

Client Subsystem (WKBCH, ODFRM and DESKT CSCIs)

The WKBCH CSCI contains the EOSView and Java DAR Tool (JDT) standalone COTS products. EOSView has it's own custom error handling.

The JDT uses the Java exception facilities to check error conditions. Exceptions are either handled in the class that produced the error or they are sent back to the calling method/class. At some point in the calling chain, the exception is either handled by the application code or it is dumped out in the stack trace (which appears in the Java Console if it's a client-side exception or in the jess.log if it's a server-side exception). The JDT's exception classes are organized in an exceptions package (jdt.aux.exceptions).

The ODFRM CSCI has no special error handling. ODFRM is a cgi application, which uses EcUtStatus to communicate errors. Functions can return an EcUtStatus object, indicating success or failure.

The DESKT CSCI consists of the User Profile Gateway. The User Profile Gateway uses EcUtStatus and exceptions. Functions can return an EcUtStatus object, indicating success or failure or throw an exception. The User Profile Gateway does not have any special error processing.

Data Management Subsystem (DDICT, V0 GTWAY, ASTGW CSCIs)

The Data Dictionary Server, V0 GTWAY and ASTGW CSCIs use two main mechanisms for error handling.

1. Return EcUtStatus values

Functions can return an EcUtStatus object, indicating success, failure or other detailed status, which corresponds to EcUtStatus:OK, EcUtStatus:FAILED and EcUtStatus:DETAILED.

For V0ToEcsGateway, the EcUtStatus values are reflected on the EDG to indicate the search or order status.

2. Exceptions

Some functions (e.g., a main function) cannot return EcUtStatus values to indicate success or failure. These functions can catch some exceptions after a try block.

All errors messages will be sent to ALOG file or debug log file.

For writing messages to the Applications Log (ALOG), the following functions are used:

EcLgLogError sends a message to the ALOG at severity level 1. For example, EcLgLogError ("Error" , 0,"Failed to get Order ID from MSS. TransactionID=%s.",

myTransactionId.data());

EcLgLogInformational sends a message to the ALOG at severity level 3. For example, EcLgLogInformational ("DmGwManagedServer", 0, "Server Shutdown");

For writing messages to the debug log, the following macros are used:

PF_STATUS writes a message at a "log level" of 1 to the debug log. For example, PF_STATUS cerr << "Unable to create inputNonAggregate " << endl;

PF_VERBOSE writes a message at a "log level" of 2 to the debug log. For example, PF_VERBOSE cerr << "~~~ Connect to Science Data Server FAILED."

PF_DEBUG writes a message at a "log level" of 3 to the debug log. For example, PF_DEBUG cerr << "~~~ Failed to get Order ID from MSS." << endl;

Product Distribution System (PDS) Subsystem

The PDS consists of two components, the Product Distribution System Input Server (PDSIS) and the Product Distribution System Stand Alone (PDSSA). Both subsystems utilize Oracle Forms as their main GUI. The GUIs display error messages to the user.

Product Distribution System (PDS) Input Server (PDSIS)

The PDSIS uses the Java exception facilities to check error conditions. Exceptions are either handled in the class that produced the error or they are sent back to the calling method/class. At some point in the calling chain, the exception is either handled by the application code or it is dumped out in the stack trace and added to the error log for the PDSIS. The error log is located under the logs/errors directory off the main PDSIS home directory.

The PDSIS also prints other messages that are located in the logs directory off the main PDSIS directory. Subdirectories contain debug, in_msg (socket in), out_msg (socket out) messages.

Exceptions that are dumped out by the PDSIS cron jobs are mailed to the PDSIS user or can alternatively be added to a log file that is created under the PDSIS account's log directory. Users do this both ways.

Product Distribution System (PDS) Stand Alone (PDSSA)

The PDSSA component uses a utility that is a part of the PDS code called log_printf to print informational, warning, and error messages from the production modules. The messages are written to the order's summary log.

Errors generated by the PDSSA cron jobs are mailed to the PDSIS user or can alternatively be added to a log file that is created under the PDSIS account's log directory. Users do this both ways.

Product Generation System (PGS) Toolkit

3.2.4 TOOLKIT Error/Status Reporting (SMF Tools)

To detect and report error and status conditions in a consistent manner standardized status messages and status codes must first be established. The method used to institute these message/code pairs is by way of the 'smfcompile' utility. But first, users need to create Status Message Files (SMFs) to contain their custom status messages and corresponding status identifiers. These identifiers take the form of user defined mnemonics that visually convey the essence of the status message. The user makes direct use of these mnemonics in their software when testing for status conditions and when interfacing with the SMF Toolkit functions. Once an SMF is completed, the smfcompile utility is run to bind the status messages and mnemonics with integral status codes. This process facilitates the run-time access of all status messages and provides for the referencing of status mnemonics within the user's code.

The status codes generated by the 'smfcompile' utility are guaranteed to be unique to ensure that there are no ambiguous status conditions, in the event that code from different Science Computing Facilities (SCFs) is merged into a single executable and/or PGE. This uniqueness is possible because "seed" values, which are different for every SMF, are used in the generation of the status codes. Typically, many SMF files are created in the course of software development; therefore many seed numbers are required. However, it is important to note that valid seed numbers can only be obtained from the Toolkit development team. Any attempt to produce SMFs from "home-grown" seed values can result in the SMFs being unusable at integration & test time.

The SDP Toolkit routines actually contain their own collection of status codes and associated status messages for describing the state of each Toolkit function. Users of the Toolkit functions should examine the return values of each tool before performing any other action. To inform a calling unit (user's software) about the exit state of a called Toolkit routine, each Toolkit function sets a status message and assigns a status code to the return value. On the basis of its interpretation of this return value, the calling unit may elect to perform some error handling. As part of this procedure, the user should either propagate the existing status code up through their calling hierarchy, or set a status code and message to represent the outcome of any local error handling attempt.

Upon detection of an error state, users are advised to report on the existing error prior to performing an error handling procedure. The content of these reports might include the following:

- A user-defined message string to convey the nature of the status condition
- A user-defined action string to indicate the next operation to be performed in response to the status condition
- A system defined string that uniquely identifies the environment in which the status condition occurred.

However, this is merely a suggestion; the users are free to define the content of the status reports to satisfy their own requirements. The method for reporting this information involves the generation of a report from the information just described and the subsequent transmission of that report to the appropriate destination(s).

The tools provided here allow for the propagation of status information within a PGE executable to facilitate a user's error handling process. They also provide the means to communicate status and error information to various monitoring authorities and event logs. Additionally, there is a tool that enables the user to specify, a priori, the action to be taken in the wake of a fatal arithmetic event. This mechanism allows users to take their own corrective measures to control an event that is terminal by default. Note that all other event conditions fall under the purview of system processing and are thereby controlled by the governing SDPS software.

Several new features have been incorporated into these tools for Toolkit 5 in order to improve their efficiency. One of those features allows for the buffering of individual status messages up to some user defined run-time limit. This should greatly reduce the amount of I/O required to access these messages. As a process proceeds to completion, new status messages are buffered as older; less used status messages become un-buffered. The goal here is to only access status messages from their run-time file when they are being referenced for the first time. The actual observed improvement depends on the degree to which a process' status messages are localized (i.e., A particular status message should ideally only be referenced within a short body of code.) and the buffer size, which is set by the user. Another feature reduces the number of replicated status messages that can appear in the status log file. This is accomplished by "compressing" duplicate messages into a count of such messages. This feature should significantly reduce the size of the status log file and contribute to its general readability.

Since each function has only one return value, every effort has been made to preserve the most important warning or error value on returning. Given that subordinate functions often have several possible returns, and different users have different priorities, it is always advisable to check the message log as well as examining the return. When totally inconsistent behavior is found in a return from a subordinate function, the returned value is PGS_E_TOOLKIT. Example: a Toolkit function passes an internally generated vector, whose length is certain to be nonzero, to a subordinate function. The lower-level function then returns a warning or error return stating that the vector is of zero length; while the higher-level function returns PGS_E_TOOLKIT. Another example: if a valid spacecraft tag is passed in, but rejected as invalid down the processing line, the error PGS_E_TOOLKIT is returned by the higher-level function. Thus return value PGS_E_TOOLKIT indicates a flaw in the software, the violation of an array boundary; a hardware, compiler, or system error; corrupted data, or some similarly serious condition that invalidates the processing.

Logging Control

PCF entry:
10114|Logging Control; 0=disable logging, 1=enable logging|1

This can be used to disable logging altogether. If logging is disabled NO message will output to any log files (although a small header is still written to the log files indicating logging for this PGE has been disabled). The Default State is for logging to be enabled.

Trace Control

PCF entry:
10115|Trace Control; 0=no trace, 1=error trace, 2=full trace|0

This can be used to specify the trace level for message logging. Tracing is a feature made possible by the addition of two SMF tools: PGS_SMF_Begin and PGS_SMF_End. Users may include these tools at the beginning and ending of their functions (respectively) to signal to the SMF system when each user defined function is entered and exited. Three levels of tracing are possible:

No Tracing

This is the Default State. No information concerning the entering or exiting of functions is recorded to the log files. No information concerning the path of a function call is recorded to the log files.

Example Log Entry:
func4():PGSTD_W_PRED_LEAPS:27652
predicted value of TAI-UTC used (actual value unavailable)

Error Tracing

If error tracing is enabled, information concerning the path of a function call is recorded to the log files any time a status message is logged to a log file. This is useful in determining where in a chain of function calls an error occurred. No information concerning the entering or exiting of functions is recorded in this state.

Example Log Entry:
```
main():
   func1():
      func2():
         func3():
            func4():PGSTD_W_PRED_LEAPS:27652
            predicted value of TAI-UTC used (actual value unavailable)
```

Full Tracing

If full tracing is enabled, a message is written to the log files each time a function is entered and exited (only those user functions with the PGS_SMF_Begin/End calls, see above). Indenting is also done to show the path of each function call.

Example Log Entry:

```
PGS_SMF_Begin: main()
  PGS_SMF_Begin: func1()
    PGS_SMF_Begin: func2()
      PGS_SMF_Begin: func3()
        PGS_SMF_Begin: func4()
          func4():PGSTD_W_PRED_LEAPS:27652
          predicted value of TAI-UTC used (actual value unavailable)
        PGS_SMF_End: func4()
PGS_SMF_End: func3()
PGS_SMF_End: func2()
PGS_SMF_End: func1()
PGS_SMF_End: main()
```

Process ID Logging

PCF entry:
10116|Process ID logging; 0=don't log PID, 1=log PID|0

This can be used to enable the tagging of log file entries with the process ID of the process from which the entry came. This is useful for PGEs that run concurrent processes that are all writing to a single log file simultaneously. If process ID logging is enabled, each log entry is tagged with the process ID of the process making the entry. This can facilitate in post-processing a log file.

Example Log Entry:
func4():PGSTD_W_PRED_LEAPS:27652 (PID=2710)
predicted value of TAI-UTC used (actual value unavailable)

Status Level Control

PCF entry:
10117|Disabled status level list (e.g., W S F)|<status level list>

This can be used to disable the logging of status codes of specific severity levels. A list of levels to be disabled should be substituted for <status level list> (e.g.: N M U). No message of a status level indicated in the list is recorded to any log. The Default State is to enable logging for all status levels.

Status Message Seed Control

PCF entry:
10118|Disabled seed list|<status code seed list>

This can be used to disable the logging of status codes generated from specific seed values. A list of seed values, the status codes derived from which should be disabled, should be substituted for <status code seed list> (e.g.: 3 5). No message derived from a seed value indicated in the list is recorded to any log. The Default State is to enable logging for all seed values.

Individual Status Code Control

PCF entry:
10119|Disabled status code list|<status code list>

This can be used to disable the logging of specific status codes. A list of status code mnemonics and/or numeric status codes should be substituted for <status code list> (e.g.: PGSTD_M_ASCII_TIME_FMT_B 678954). Note that using mnemonics can disable only Toolkit status codes. To disable a user generated status code a numeric status code must be used. No messages whose status codes or mnemonics included in the list are recorded to any log file. The Default State is to enable logging for all status codes.

Generating Runtime E-Mail Messages

A PGE may be configured to automatically generate and send e-mail messages during run-time when specific user defined status codes are logged. This is done by assigning an e-mail action to a given user defined status code.

An e-mail action is an SMF code with the special status level of "C" and a mnemonic that begins with the characters "PGSEMAIL" (the rest of the mnemonic may contain any other valid mnemonic characters), for example:

PGS_*C_PGSEMAIL*_SEND_EMAIL
ASTER_*C_PGSEMAIL*_ALERT
MODIS_*C_PGSEMAIL*_ERROR

An e-mail message is generated anytime a user defined status code with an associated e-mail action is logged via the SMF logging routines. The contents (body) of these messages are the text (message) associated with the user defined status code. The subject of these messages is the mnemonic associated with the user defined status code. The list of recipients is defined in the e-mail action definition.

Example:
In a user defined status message file the following status code mnemonic label and e-mail action mnemonic label have been defined (the e-mail action is associated with the status code via the "::" syntax):

```
MODIS_E_PGE_INIT_FAILED   The PGE failed to initialize.
                          ::MODIS_C_PGSEMAIL_NOTIFY
MODIS_C_PGSEMAIL_NOTIFY   john@modis.org, sue@modis.org
```

The following lines appear in a C source code file:

```
    returnStatus = initializePGE();
    if (returnStatus == MODIS_E_PGE_INIT_FAILED)
```

```
{
    PGS_SMF_SetStaticMsg(returnStatus, “main()”);
    exit(1);
}
```

At run-time, if the returned status code from the function initializePGE() has the value defined by MODIS_E_PGE_INIT_FAILED, this status is logged via the SMF function PGS_SMF_SetStaticMsg(), and because this status code has an e-mail action associated with it, an e-mail message is generated.

The e-mail message is sent to: sue@modis.org and john@modis.org
The subject field of the e-mail message is: MODIS_E_PGE_INIT_FAILED
The text of the e-mail message is: The PGE failed to initialize.

Note:
This functionality is disabled at the DAACs.

3.2.5 Memory Management

Object-oriented modeling and design is a new way of thinking about problems using models organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity. Objected-oriented models are useful for understanding problems, communicating with application experts, modeling enterprises, preparing documentation and designing programs and databases.[1]

Superficially, the term "object-oriented,” means that we organize software as a collection of discrete objects that incorporate both data structure and behavior. This is in contrast to conventional programming in which data structure and behavior are only loosely connected. There is some dispute about exactly what characteristics are required by an object-oriented approach, but generally include four aspects: identity, classification, polymorphism and inheritance.[1]

Identity means that data is quantized into discrete, distinguishable entities called *objects*. A paragraph in my document, a window on my workstation and a white queen in a chess game are examples of objects. Objects can be concrete, such as a file, or conceptual, such as a *scheduling policy* in a multi-processing operating system. Each object has its own inherent identity. In other words, two objects are distinct even if all their attribute values (such as name and size) are identical.[1]

In the real world an object simply exists, but within a programming language each object has a unique *handle* by which it can be uniquely referenced. The handle may be implemented in various ways, such as an address, array index or unique value of an attribute. Object references are uniform and independent of the contents of the objects, permitting mixed collections of objects to be created, such as a file system directory that contains both files and sub-directories.[1]

Classification means that objects with the same data structure (attributes) and behavior (operations) are grouped into a *class*. Paragraph, Window, and ChessPiece are examples of classes. A *class* is an abstraction that describes properties important to an application and ignores the rest. Any choice of classes is arbitrary and depends on the application.[1]

Each class describes a possibly infinite set of individual objects. Each object is said to be an instance of its class. Each instance of the class has its own value for each attribute but shares the attribute names and operations with other instances of the class. An object contains an implicit reference to its own class: it "knows what kind of a thing it is."[1]

Polymorphism means that the same operation may behave differently on different classes. The *move* operation, for example, may behave differently on the *Window* and *ChessPiece* classes. An *operation* is an action or transformation that an object performs or is subject to. *Right justify, display* and *move* are examples of operations. A specific implementation of an operation by a certain class is called a *method*. Because an object-oriented operator is polymorphic, it may have more than one method implementing it.[1]

In the real world, an operation is simply an abstraction of analogous behavior across different kinds of objects. Each object "knows how" to perform its own operations. In an object-oriented programming language, however, the language automatically selects the correct method to implement an operation based on the name of the operation and the class of the object being operated on. The user of an operation need not be aware of how many methods exist to implement a given polymorphic operation. New classes can be added without changing existing code, provided methods are provided for each applicable operation on the new classes.[1]

Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship. A class can be defined broadly and then refined into successively finer *subclasses.* Each sub-class incorporates, or *inherits* all the properties of its *super-class* and adds its own unique properties. The properties of the superclass need not be repeated. For example, *ScrollingWindow* and *FixedWindow* are subclasses of *Window*. Both subclasses inherit the properties of *Window*, such as a visible region on the screen.[1]

The ECS is a large, complex data storage and retrieval system used to store and retrieve large amounts of science and science-related data. The system was designed using an object oriented design approach. With so many objects and the sizes of some of them, it is necessary to have some insight into the amount of memory being utilized within the ECS. The information about to be presented is a brief look at the memory management of the "key" (top ten utilized) objects within the ECS subsystems.

In this object oriented system design, objects are created and used via classes throughout the system to help perform the functions and meet the needs of the system. The objects for the ECS are very numerous, sometimes very large and cannot be provided in their entirety at this time. However, presented in the table below are the "key" objects for this subsystem and how they are created, passed and deleted within the ECS.

Introduction to memory management approaches and memory usage table

Good memory management in some applications is both important and requires significant planning and development time. Many important ECS applications are large, long running, multi-threaded, heavy memory users and therefore are prime candidates for improved memory management.

Improper memory management can result in memory leaks, fast memory usage growth or large application footprints and random crashes. ECS servers are periodically purified for memory

leaks and there is a history of progress in this area. Similar work should be expected to continue as development and maintenance continues.

Long running server like applications that are free from memory leaks can nonetheless have significant memory and Central Processing Unit (CPU) usage performance degradation. A common culprit is heap fragmentation. The repeated allocation and deallocation of memory (such as with the new and delete operators of C++) can result in a large number of unusable free blocks of memory. They are free blocks but are interspersed with non-free blocks. They become unusable since they are not contiguous (fragmented) and as time goes by, it becomes harder and harder for the OS to service requests for more memory. Such situations even lead to crashes of other, non-offending applications running in the same box.

There are strategies, tools and software to avoid both memory leaks and fragmentation. This includes but is not limited to:

1. Periodic application of purification software (already an ECS practice)
2. Software design, which uses dynamic memory as little as possible, such as automatic storage or COTS data structures
3. Class-level memory management to allocate large chunks of memory instead of one class instance at a time ("Effective C++" by Scott Meyers and "Advanced C++" by James Coplien address this technique)
4. Non-class level memory pools and
5. COTS heap manager

The table below is provided in case further memory management improvements are desired. Given operator or field input of seemingly inefficient memory or CPU usage, this table can be used to help target specific ECS subsystems, servers and frameworks or classes for improvement. It can be decided to apply some of the approaches at one level (e.g., on one guinea pig server or class) or perhaps experiment with changing the entire ECS C++ system with the use of a COTS heap manager. In any case, a great deal of planning and manpower is required.

Table 3.2.5-1. Memory Management Improvements (1 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
DSS - SDSRV	EcDsScienceDataServer	DsSrSession	Manages threads to execute requests associated with the Working Collection.	EcDsScienceDataServer (class: DsSrConnectionMaker)	DsSrManagedServer	DsSrManagedServer	One per DsSrWorkingCollection	Object is deleted when the Science Data Server goes down.
	EcDsScienceDataServer	DsSrWorkingCollection	A "shopping basket" class containing a vector of ESDTs used to execute requests.	EcDsScienceDataServer (class: DsSrSession class).	EcDsScienceDataServer (class: DsSrWarmStartManager)	Deleted when its corresponding DsSrSession object is deleted.	One per DsSrSession object.	
	EcDsScienceDataServer	DsSrWarmStartManager	Singleton class controlling processing of asynchronous requests.	EcDsScienceDataServer (class: DsSrManagedServer)	not passed	EcDsScienceDataServer	Either 1 instance or no instances for all of SDSRV.	Static singleton class. Object is deleted when the Science Data Server goes down.

Table 3.2.5-1. Memory Management Improvements (2 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	Clients, EcDsScienceDataServer	DsShRequestReal	This class provides a server interface to the server's request distributed object. It inherits from the DCE-generated server request class, and adds functions to provide stronger type checking.	Clients, EcDsScience DataServer	EcDsScienceDataServer or other servers related	When request is finished or server goes down	1 per client request.	This class communicates between clients and EcDsScienceDataServer.

Table 3.2.5-1. Memory Management Improvements (3 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	Clients and EcDsScienceDataServer	GlParameterList	This library is used by many subsystems to provide a general-purpose list object for storing various scalar and complex data types.	Clients and EcDsScience DataServer	EcDsScienceDataServer and EcDsDistributionServer	Clients	Could have many in each request	To group one or more GlParameter derived classes that store the various parameter types required building commands. Any Gl type, including embedded GlParameterLists can be inserted into a GlParameterList, making it recursive in design.

Table 3.2.5-1. Memory Management Improvements (4 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsScienceDataServer	DsGeESDT	Inherit from the public class - provides basic ESDT functionality.	EcDsScience DataServer	EcDsDistributionServer, EcDsStRequestManagerServer and other related applications	EcDsScience DataServer and other related applications	1 per granule	This class provides functionality common to all SDSRV data types.
	DsSrManaged Server::DsSrStart()	DsBtSbsrvNotifier	This class is used to facilitate communication between the SDSRV and SBSRV through an event queue.	EcDsScience DataServer [class::method = DsSrManaged Server::DsSrStart()]	not passed	EcDsScience DataServer [class::method = DsSrManagedServer::DsSrStart()]	Configurable	Only one instance per session is created. The object goes away when the server goes down.

Table 3.2.5-1. Memory Management Improvements (5 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsScienceDataServer [function = DsGeESDT::Insert()]	DsMdMetadata	This class is used as a container class for metadata.	EcDsScience DataServer [function = DsGeESDT::Insert()]	not passed	See comments/remarks column.	1 per DsGeESDT	When this object is instantiated, it uses the local memory manager. The object can be saved to the database if the user is executing an insert.
	EcDsScienceDataServer [class::method= DsSrGenCatalogPool:::DsSrGenCatalogPool()]	DsMdCatalog	This class is used to manage catalog pools.	DsSrGenCatalogPool:::DsSrGenCatalogPool()	not passed	EcDsScience DataServer	Depends on the configured pool size.	There are three default pools for catalogs: SEARCH, INSERT and DEFAULT. The object goes away when the server goes down.

Table 3.2.5-1. Memory Management Improvements (6 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsScienceDataServer	DsDbInterface	Database (Sybase) interface class to encapsulate database related services such as: connect, execute, fetch result	EcDsScience DataServer	not passed	When EcDsScience DataServer is down	2 per DsMsCatalog	User can connect to DB, execute SQL statements, verify connection states and disconnect from the database.
	Clients	DsClESDTReferenceCollector	Provides the primary interaction mechanism for client software.	Clients	EcDsScienceDataServer	EcDsSdsrvTest or EcDsTsClientDriver or when clients go down	1 per client connection	As its name implies, it is a collector of the DsClESDTReference object referred to as the clients "working collection", which is populated with the results of service requests such as "Acquire", "Insert", "Search."

Table 3.2.5-1. Memory Management Improvements (7 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
PDS	pdsis.jar – start script EcPdPDSISServer	EcPdPDSISSserver	Creates listening socket for order requests. Reads requests and enters them into the PDSIS system	pdsis.jar – start script EcPdPDSISServer	Not Passed	When server stopped	1	Validates incoming ODL data and passes back to sender order status.
OM	EcOmOrderManager	OmSrDispatchQueue	Queues all the requests up in different queues according to their destinations and functionalities	EcOmOrderManager	Not Passed	When server stopped	Four (4) queues per EcOmOrderManager	

Table 3.2.5-1. Memory Management Improvements (8 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
DSS - STMGT	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer	DsStDictionary	Maps a string name to a string value	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui	not passed not passed not passed not passed not passed not passed	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui	Used in Singleton DsStStProcTable	Object is deleted when the server goes down.

Table 3.2.5-1. Memory Management Improvements (9 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
DSS - STMGT	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui	DsStStProcTable	A singleton used to create a dictionary that associates a stored procedure name with a list of parameters in the order that they appear in the stored procedure declaration.	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui	not passed not passed not passed not passed not passed not passed	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui	Singleton	Object is deleted when the server goes down.

Table 3.2.5-1. Memory Management Improvements (10 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui EcDsDistributionServer EcInReqMgr EcInGran	DsStRemoteTransaction	Handles the remote transactions for stored procedure call. It converts the request in a format, which can be passed across a network interface.	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui EcDsDistributionServer EcInReqMgr EcInGran	not passed not passed not passed not passed not passed not passed not passed not passed not passed	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui EcDsDistributionServer EcInReqMgr EcInGran	4 per request (client request and result, server request and result)	

Table 3.2.5-1. Memory Management Improvements (11 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui EcDsDistributionServer EcInReqMgr EcInGran	DsStRemoteSP	Parent class of DsStRemoteTransaction	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui EcDsDistributionServer EcInReqMgr EcInGran	not passed not passed not passed not passed not passed not passed not passed not passed not passed	EcDsStRequestManagerServer EcDsStCacheManagerServer EcDsStStagingDiskServer EcDsStArchiveServer EcDsStFtpServer EcDsStmgtGui EcDsDistributionServer EcInReqMgr EcInGran	Multiple times per request	
	DsStRequestManagerServer	DsStRequest	Describes all possible states of a request	DsStRequestManagerServer	not passed	DsStRequestManagerServer	1 per request	

Table 3.2.5-1. Memory Management Improvements (12 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	DsStRequestManagerServer	DsStRequestQueue	Provides queuing mechanism for DsStRequest objects	DsStRequest ManagerServer	not passed	DsStRequest ManagerServer	Singleton	
	EcDsScienceDataServer(DLL), EcDsDistributionServer, EcDsStArchiveServer	DsStFileParameters	Data structure to maintain the file related parameters	EcDsScience DataServer (DLL), EcDsDistributionServer, EcDsStArchiveServer	not passed not passed not passed	EcDsScience DataServer (DLL), EcDsDistributionServer, EcDsStArchiveServer	Multiples times per request within Archive server and Archive client	

Table 3.2.5-1. Memory Management Improvements (13 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
DSS - STMGT	EcDsStCache ManagerServer, EcDsStStaging DiskServer, DsStArchiveServer	DsStCopyService	File I/O operation for Copy service	EcDsStCache ManagerServer, EcDsStStagingDiskServer, DsStArchiveServer	not passed not passed not passed	EcDsStCacheManagerServer, EcDsStStagingDiskServer, DsStArchiveServer	One per CacheManagerServiceThread, One per StagingDiskServiceThread, One per ArchiveWriteThread	The item should be the DsStCopyService class rather than the DsStCopyService::Copy function.
	EcDsStCache ManagerServer, EcDsStStaging DiskServer, DsStArchiveServer	DsStFtpService::FtpCopy( )	File I/O operation for Copy service	EcDsStCache ManagerServer, EcDsStStagingDiskServer, DsStArchiveServer	not passed not passed not passed	EcDsStCacheManagerServer, EcDsStStagingDiskServer, DsStArchiveServer	One per CacheManagerServiceThread, One per StagingDiskServiceThread, One per ArchiveWriteThread	

Table 3.2.5-1. Memory Management Improvements (14 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsStCache ManagerServer, EcDsStStaging DiskServer, DsStArchiveS erver	DsStFtpService::F tpRetr( )	File I/O operation for Copy service	EcDsStCache ManagerServer, EcDsStStagingDiskServer, DsStArchiveS erver	not passed not passed not passed	EcDsStCach eManagerSer ver, EcDsStStagi ngDiskServe r, DsStArchive Server	One per CacheMan agerServic eThread, One per StagingDis kServiceT hread, One per ArchiveWr iteThread	
DSS - DDIST	EcDsDistributi onServer	DsDdMedia	Contains media drivers and request level media information, like media type.	EcDsDistribut ionServer	not passed	EcDsDistribu tionServer	Created once per DDIST request.	
	EcDsDistributi onServer	DsDdDistRequest S	Contains request level information, like State and orderID	EcDsDistribut ionServer	not passed	EcDsDistribu tionServer	Created once per DDIST request.	
	EcDsDistributi onServer	DsDdDistListS	Contains pointers to granule and file information for the request	EcDsDistribut ionServer	not passed	EcDsDistribu tionServer	Created once per DDIST request.	

Table 3.2.5-1. Memory Management Improvements (15 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsDistributionServer	DsDdMediaDist	Contains packaging information and has a one-to-one correspondence to physical media for the request. Note: requests can be sufficiently large to span more than one physical media.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created at least once per each DsDdMedia	
	EcDsDistributionServer	DsDdGranuleS	Contains granuleUR information.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created for each granule for each request.	
	EcDsDistributionServer	DsDdDistFileS	Contains file information, like file name & file size.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created for each file (archive data file, metadata file, etc.) in each granule	

Table 3.2.5-1. Memory Management Improvements (16 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	Created in most functions and most object constructors	RWCString	The rogue-wave string class used through all classes and functions.	Created in most functions and most object constructors.	An extensive list of functions.	An extensive list of functions.	Created in most functions and most object constructors.	
	Created in most functions.	DsDdLog	Utility for simplifying logging.	Created in most functions.	EcDsDistributionServer	EcDsDistributionServer	Created in most functions.	
	EcDsDistributionServer	DsDdActiveQueue	A Queue of requests that are in Active, Staging, and Transferring states.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	
	EcDsDistributionServer	DsDdBaseQueue	Parent class of the remaining queues	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	
	EcDsDistributionServer	DsDdConfiguration	Utility class for reading configuration information.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	

Table 3.2.5-1. Memory Management Improvements (17 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsDistributionServer	DsDdDoneQueue	A Queue of requests in the Shipped, Failed or Cancelled states.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	
	EcDsDistributionServer	DsDdHoldQueue	A Queue of requests that are in one of the suspended states.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	
	EcDsDistributionServer	DsDdPriorityQueue	A Priority Queue of requests that are pending. Uses DsDdThreadPool class in the process of selecting the next request for processing.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	
	EcDsDistributionServer	DsDdReadyToShipQueue	A Queue of hard media requests waiting on the operator for selection to enter the Shipped state.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	

Table 3.2.5-1. Memory Management Improvements (18 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDsDistributionServer	DsDdRequestLists	List of all DDIST requests	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	
	EcDsDistributionServer	DsDdScheduler	Class, which wakes up worker threads and assigns them to requests as selected by the DsDdPriorityQueue.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once (singleton class)	
	EcDsDistributionServer	DsDdThreadPool	Uses thread pools DB tables and procedures to select next request for processing, given request information and constraints.	EcDsDistributionServer	not passed	EcDsDistributionServer	Created once per each thread pool name (usually 5 - 50)	

Table 3.2.5-1. Memory Management Improvements (19 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
DMS	DMS	GlParameterList:: DeepAssign( )	Generic copy method for ECS composite class	DMS	SDSRV (CSCI), PLANG (CSCI), Registry (CSC)	Where appropriate	Extensive	DMS code compensates for native leak by client side action
	DmGwV0EcsRequestReciever	DmGwInventoryRequest	Handler for V0 Inventory Searches	DmGwV0EcsRequestReciever	not passed	DmGwV0EcsRequestReciever	1 per thread	No significant leaks
	DmGwV0EcsRequestReciever	DmGwBrowseRequest	Handler for V0 Browse Requests	DmGwV0EcsRequestReciever	not passed	DmGwV0EcsRequestReciever	1 per thread	No significant leaks, but for integrated browse may hold large amount of memory while active
	DmGwV0EcsRequestReciever	DmGwProductRequest	Handler for V0 Product Requests	DmGwV0EcsRequestReciever	not passed	DmGwV0EcsRequestReciever	1 per thread	No significant leaks
	EcDmV0ToEcsGateway	DmGwSpecializedCriteria	Representation of V0 SPECIALIZED_CRITERIA element	EcDmV0ToEcsGateway	not passed	Where appropriate	Extensive	Recursive class, potential for large memory usage. No significant leaks.

Table 3.2.5-1. Memory Management Improvements (20 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	DmGwV0EcsRequestReciever	DmGwDirectoryRequest	Handler for V0 Directory Searches	DmGwV0EcsRequestReciever	not passed	DmGwV0EcsRequestReciever	1 per thread	No significant leaks
	DmGwInventoryRequest	DmGwGranuleLevelSearch	SDSRV search client-side search object	DmGwInventoryRequest	not passed	DmGwInventoryRequest	1 per data set searched	No significant leaks
	DmGwManagedServer	DmGwRequestReceiver	Listener / Dispatch class for EcDmV0ToEcsGateway	DmGwManagedServer	not passed	DmGwManagerdServer	1 per session	No significant leaks
	DmGwDataset	EcRgDistOptions	Client class for distribution options provision	DmGwDataset	not passed	DmGwDataset	1 per search	No significant leaks
	DmGwDataset	EcRgSubsetOptions	Client class for subset options provision	DmGwDataset	not passed	DmGwDataset	1 per search	No significant leaks

Table 3.2.5-1. Memory Management Improvements (21 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
DPS	EcDpPrEM EcDpPrDeletion EcPlPREditor_IF EcPlOdMgr	DpPrDSSInterface	Interface to SDSRV	EcDpPrEM EcDpPrDeletion EcPlPREditor_IF EcPlOdMgr	not passed	EcDpPrEM EcDpPrDeletion EcPlPREditor_IF EcPlOdMgr	# granules x DPR # interim files x DPR # input granules x DPR # input granules x OD DPR	
	EcDpPrEM	DpPrDataManager	Manages acquires and inserts of granules from/to SDSRV	EcDpPrEM	not passed	EcDpPrEM	# granules x DPR	
	EcDpPrEM	DpPrExecutionManager	Supervisory program for DPR execution	EcDpPrEM	not passed	EcDpPrEM	2 x DPR	
	EcDpPrEM	DpPrResourceManager	Manages disk allocation for files and CPUs for DPRs	EcDpPrEM	not passed	EcDpPrEM	(# files + 2) x DPR	

Table 3.2.5-1. Memory Management Improvements (22 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcDpPrJobMgmt EcDpPrEM	DpPrScheduler	Manages DPR construction in AutoSys and scheduling on computers	EcDpPrJobMgmt	not passed	EcDpPrJobMgmt	1 x DPR and until server is brought down	
	EcDpPrEM	DpPrPcf	Constructs process control file	EcDpPrEM	not passed	EcDpPrEM	1x DPR	
	EcDpPrJobMgmt	DpPrCotsManager	Interface to AutoSys	EcDpPrJobMgmt	not passed	EcDpPrJobMgmt	2 x DPR	
	EcDpPrJobMgmt	DpPrJIL	Interface to AutoSys	EcDpPrJobMgmt	not passed	EcDpPrJobMgmt	1 x DPR	
	EcDpPrEM	DpPrPge	Manages acquire of PGE	EcDpPrEM	not passed	EcDpPrEM	1 x DPR	
	EcDpPrEM	DpPrFile	Helper class for DprPrDataManagement	EcDpPrEM	not passed	EcDpPrEM	# files x DPR	
	EcDpPrDeletion	DpDeletionServer	Removes PDPS files that are no longer used	EcDpPrDeletion	not passed	EcDpPrDeletion	until server is brought down	
	EcDpPrDeletionClient	DpDeletionProxy	Responsible for identifying files to be deleted	EcDpPrDeletionClient	not passed	EcDpPrDeletionClient	2 x day	

Table 3.2.5-1. Memory Management Improvements (23 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
PLS	EcPlPREditor	PlDpr	Data Processing Request class	EcPlPREditor	not passed	EcPlPREditor	1 per DPR	Created as a member of a static pool which is deleted when PREditor is brought down
	EcPlPREditor	PlUserParameters	PGE Processing Parameters class	EcPlPREditor	not passed	EcPlPREditor	Multiple per PGE	
	EcPlPREditor, EcPlOdMgr, EcPlSubMgr	PlDprData	Record that relates each input/output granule with a Data Processing Request	EcPlPREditor, EcPlOdMgr, EcPlSubMgr	not passed not passed not passed	EcPlPREditor, EcPlOdMgr, EcPlSubMgr	1 per unavailable primary/alternate input	
	EcPlPREditor, EcPlOdMgr	PlDataGranule	Record for each input/output granule	EcPlPREditor, EcPlOdMgr	not passed not passed	EcPlPREditor, EcPlOdMgr	1 per input/output granule	
	EcPlPREditor, EcPlOdMgr	PlPge	Record of PDPS PGE information	EcPlPREditor, EcPlOdMgr	not passed not passed	EcPlPREditor, EcPlOdMgr	1 per Page	Created as a member of a static pool which is deleted when PREditor or OdMgr are brought down

Table 3.2.5-1. Memory Management Improvements (24 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcPlOdMgr	PlHighLevelOrder	Record of ASTER OnDemand HighLevel Order	EcPlOdMgr	not passed	EcPlOdMgr	1 per HighLevel order	Deleted after each ASTER OnDemand order is processed
	EcPlPREditor, EcPlOdMgr	PlDataScheduled	Record of Data Scheduled type PGE	EcPlPREditor, EcPlOdMgr	not passed not passed	EcPlPREditor, EcPlOdMgr	1 per Data Scheduled PGE	Deleted after PGE information is collected
	EcPlPREditor, EcPlOdMgr	PlDataTypeReq	Record of ESDT of Data Processing Request input	EcPlPREditor, EcPlOdMgr	not passed not passed	EcPlPREditor, EcPlOdMgr		
	EcPlPREditor, EcPlOdMgr	PlMetadataChecks	Record of required metadata checks for PGE inputs	EcPlPREditor, EcPlOdMgr	not passed not passed	EcPlPREditor, EcPlOdMgr	1 per PGE input if metadata checks are required	Deleted when the collection is destroyed
	EcPlPREditor	PlUserParameters	Record of user defined processing parameter values	EcPlPREditor	not passed	EcPlPREditor	1 per Page, if defined	Deleted when the collection is destroyed
	EcPlPREditor, EcPlOdMgr	PlTimeScheduled	Class that represents a Time Scheduled type PGE	EcPlPREditor, EcPlOdMgr	not passed not passed	EcPlPREditor, EcPlOdMgr	1 per Time Scheduled PGE	Deleted after PGE information is collected

Table 3.2.5-1. Memory Management Improvements (25 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcPlOdMgr	PlStandingOrderGranules	Collection of ASTER OnDemand input granules with DAR Ids associated with user requests	EcPlOdMgr	not passed	EcPlOdMgr	1 per Standing Order with a matching DAR ID	Deleted after all relevant OnDemand orders are processed
	EcPlPREditor, EcPlOdMgr	PlSnapshotScheduled	Class that represents a Snapshot Scheduled type PGE	EcPlPREditor, EcPlOdMgr	not passed not passed	EcPlPREditor, EcPlOdMgr	1 per Snapshot Scheduled PGE	Deleted after PGE information is collected
	EcPlPREditor	PlRoutineArrival	Class that represents an input data ESDT that is ingested at regular, predictable time intervals	EcPlPREditor	not passed	EcPlPREditor	1 per routinely arriving input granule	Deleted after Data Processing Request information is collected
	EcPlPREditor, EcPlOdMgr	PlOutputYield	Class that represents Data Processing Request outputs ESDTs	EcPlPREditor, EcPlOdMgr	not passed not passed	EcPlPREditor, EcPlOdMgr		

Table 3.2.5-1. Memory Management Improvements (26 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
CSS	EcSbSubServer	GlParameterList	A class that collects the general parameters of ECS.	EcSbSubServer	EcSbCl	EcSbSubServer	9	
	EcSbSubServer			EcSbSubServer	not passed	EcSbSubServer	66	
	EcSbGui			EcSbGui	not passed	EcSbGui	19	
	EcCsEmailParser			EcCsEmailParser	not passed	EcCsEmailParser	9	
	EcCsMojoGateway			EcCsMojoGateway	not passed	EcCsMojoGateway	29	
	EcCsMtMGateway			EcCsMtMGateway	not passed	EcCsMtMGateway	31	
	EcCsRegistry			EcCsRegistry	not passed	EcCsRegistry	26	

Table 3.2.5-1. Memory Management Improvements (27 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcSbSubServer EcCsRegistry LoadingTool	RWDBMemTable	A Rogue Wave DB class that is a table of data residing in the program memory. After construction, an RWDBMemTable is no longer associated with a table in the database. An application can modify the data in an RWDBMemTable, but the changes are not propagated back to the database.	EcSbSubServer EcCsRegistry LoadingTool	not passed not passed not passed	EcSbSubServer EcCsRegistry LoadingTool	3 8 2	

Table 3.2.5-1. Memory Management Improvements (28 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcSeLoginProg EcSbSubServer LoadingTool EcCsRegistry	RWDBResult	A Rogue Wave DB class that represents a sequence of results whenever a database operation can potentially produce multiple SQL table expressions. Triggers that can cause results to be generated as a result of an INSERT, DELETE, or UPDATE statement.	EcSeLoginProg EcSbSubServer LoadingTool EcCsRegistry	not passed not passed not passed not passed	EcSeLoginProg EcSbSubServer LoadingTool EcCsRegistry	2 6 15 3	
	EcSeLoginProg EcSbSubServer EcCsRegistry LoadingTool EcCsIdNameServer	RWDBReader	A Rogue Wave DB class that provides row-by-row access to tabular data.	EcSeLoginProg EcSbSubServer EcCsRegistry LoadingTool EcCsIdName Server	not passed not passed not passed not passed not passed	EcSeLoginProg EcSbSubServer EcCsRegistry LoadingTool EcCsIdName Server	13 13 8 5 2	

Table 3.2.5-1. Memory Management Improvements (29 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcSbGui	EcClEventCollector	This class provides a collection mechanism for retrieving and manipulating multiple events.	EcSbGui	not passed	EcSbGui	1	
	EcSbGui	EcClSubscriptionCollector	This class provides a collection mechanism for retrieving and manipulating multiple subscriptions	EcSbGui	not passed	EcSbGui	1	

Table 3.2.5-1. Memory Management Improvements (30 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcCsMtMGateway EcCsMojoGateway EcCsEmailParser EcSbSubServer	DsClESDTReferenceCollector	This class provides the primary interaction mechanism for client software. This class contains the specialized functions pertaining to management of state (the working collection on the server side) by mimicking that state on the client machine.	EcCsMtMGateway EcCsMojoGateway EcCsEmailParser EcSbSubServer	not passed not passed not passed not passed	EcCsMtMGateway EcCsMojoGateway EcCsEmailParser EcSbSubServer	1 1 1 1	

Table 3.2.5-1. Memory Management Improvements (31 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
MSS	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	RWDBMemTable	A Rogue Wave DB class that is a table of data residing in the program memory. After construction, an RWDBMemTable is no longer associated with a table in the database. An application can modify the data in an RWDBMemTable, but the changes are not propagated back to the database.	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	not passed not passed not passed	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	2 1 6	

Table 3.2.5-1. Memory Management Improvements (32 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	RWDBResult	A Rogue Wave DB class that represents a sequence of results whenever a database operation may potentially produce multiple SQL table expressions. Triggers that can cause results to be generated as a result of an INSERT, DELETE, or UPDATE statement.	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	not passed not passed not passed	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	13 10 6	
	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	RWDBReader	A Rogue Wave DB class that provides row-by-row access to tabular data.	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	not passed not passed not passed	EcMsAcOrderSrvr EcMsAcRegUserSrvr MsCsSurveyMgrServer	16 13 29	

Table 3.2.5-1. Memory Management Improvements (33 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
INS	EcInGUI, EcInInitPasswd	CsSeCryptoDes:: DesEncrypt	Used to encrypt passwords.	EcInGUI, EcInInitPasswd	not passed	EcInGUI	Three instances per update to the InExternalDataProviderInfo table (EcInGUI). One instance for EcInInitPasswd.	EcInInitPasswd does not delete the instance, but it is a test driver. The amount of memory allocated is the size of the encrypted password.
	EcInGUI, EcInPolling, EcInReqMgr	CsSeCryptoDes:: DesDecrypt	Used to decrypt passwords.	EcInGUI, EcInPolling, EcInReqMgr	not passed not passed not passed	EcInGUI, EcInPolling, EcInReqMgr	Three instances per retrieval from the InExternalDataProviderInfo table.	The amount of memory allocated is the size of the encrypted password.

Table 3.2.5-1. Memory Management Improvements (34 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	InDataTypeTemplate	Used to access the InDataTypeTemplate table in the Ingest database. This table contains information about each data type that can be ingested.	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	not passed not passed not passed not passed	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	One instance	The memory is deallocated when the server comes down.
	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	InEDPAddressMap	Used to access the InEDPAddressMap table in the Ingest database. This table contains IP addresses, which can be mapped to external data provider names.	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	not passed not passed not passed not passed	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	One instance	The memory is deallocated when the server comes down.

Table 3.2.5-1. Memory Management Improvements (35 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	InCurrentDataTypeMap	Used to access the InCurrentDataTypeMap table in the Ingest database. This table contains the default version id for each data type that can be ingested.	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	not passed not passed not passed not passed	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	One instance	The memory is deallocated when the server comes down.
	EcInGUI	InValRequestState	Used to access the InValRequestState table in the Ingest database. This table contains the valid request states.	EcInGUI	not passed	EcInGUI	One instance	
	EcInGUI	InValDataGranuleState	Used to access the InValDataGranuleState table in the Ingest database. This table contains the valid granule states.	EcInGUI	not passed	EcInGUI	One instance	

Table 3.2.5-1. Memory Management Improvements (36 of 36)

Subsystem Name	Executable Name (M)	Key Classes	Description (M)	Where Created? (Executable/ process name) (M)	Passed To (Executable /process name)	Where Deleted? (Process name) (M)	Number of Instances (Example – 1 per granule)	Comments/ Remarks (Items of special interest. Example - Size per instantiation, never "deleted", etc.)
	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	InConfig	Used to store configuration parameters for Ingest.	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	not passed not passed not passed not passed	EcInGUI, EcInGran, EcInPolling, EcInReqMgr	One instance	The memory is deallocated when the server comes down.
	EcInReqMgr	InGranuleServers Queue	Used to access the Ingest granule queue. EcInReqMgr uses the granule queue for dispatching granules to each EcInGran instance.	EcInReqMgr	not passed	EcInReqMgr	One instance	The memory is deallocated when the server comes down.
CLS	Not Applicable							
Toolkit	Not Applicable							

3.3 ESDT Handling Scenario

3.3.1 Scenario Description

This scenario shows how Earth Science Data Types (ESDTs) are handled in the ECS system. This scenario is divided into a thread for the installation process, a thread for the update process, and threads for the Import and Export of ESDTs. A detailed description of these threads is provided below.

3.3.2 Scenario Preconditions

The preconditions for each thread are shown in the thread sections for this scenario.

3.3.3 Scenario Partitions

The ESDT Handling Scenario has been partitioned into the following threads:

- **Install ESDT** (Thread A) -- This thread shows how the ESDTs are installed in the ECS system (see Section 3.3.4).
- **Update ESDT** (Thread B) -- This thread shows how the ESDTs are updated in the ECS system (see Section 3.3.5).
- **Install Reference Descriptor** (Thread C) – (Deleted).
- **Update Reference Descriptor** (Thread D) – (Deleted).
- **Import ESDTs** (Thread E) – This thread shows how ESDTs are imported into the ECS system from other protocols, namely ASTER GDS (see section 3.3.8).
- **Export ESDTs (**Thread F) – This thread shows how ESDTs are exported from the ECS system to other protocols, namely V0-IMS and ASTER GDS (see section 3.3.9).

3.3.4 Install ESDT Thread

This thread shows how Earth Science Data Types (ESDTs) are installed in the ECS system. The purpose is to have ESDT data available in various applications for utilization with advertising, archiving, and subscribing to designated events. The installation of the ESDT requires a descriptor file, and a Dynamic Link Library file (DLL). The ESDT descriptor file contains Collection level and Inventory level metadata and data services information. The Dynamic Link Library file contains the services that are available for the ESDT.

To accomplish the install, the DAAC operator first identifies the ESDT. The DSS Science Data Server (SDSRV) sends applicable parts of the ESDT to the Data Dictionary (DDICT) Server and the Subscription Server (SBSRV). The Science Data Server also stores the ESDT information in its own database.

The ESDTs include data for specific instruments on each mission, external ancillary data, and System data, which includes FDD (orbit and attitude) data.

Install ESDT Thread Preconditions

- The ESDT is approved for installation.
- The DAAC Operator knows where the descriptor and the Dynamic Link Library (DLL) for the ESDT are located.
- Any file space needed for the ESDT or handling the ESDT is not provided for explicitly in this scenario. File space is handled as needed by the data servers working with the ESDTs.

Install ESDT Interaction Diagram - Domain View

Figure 3.3.4.1-1 depicts the Install ESDT Interaction - Domain View

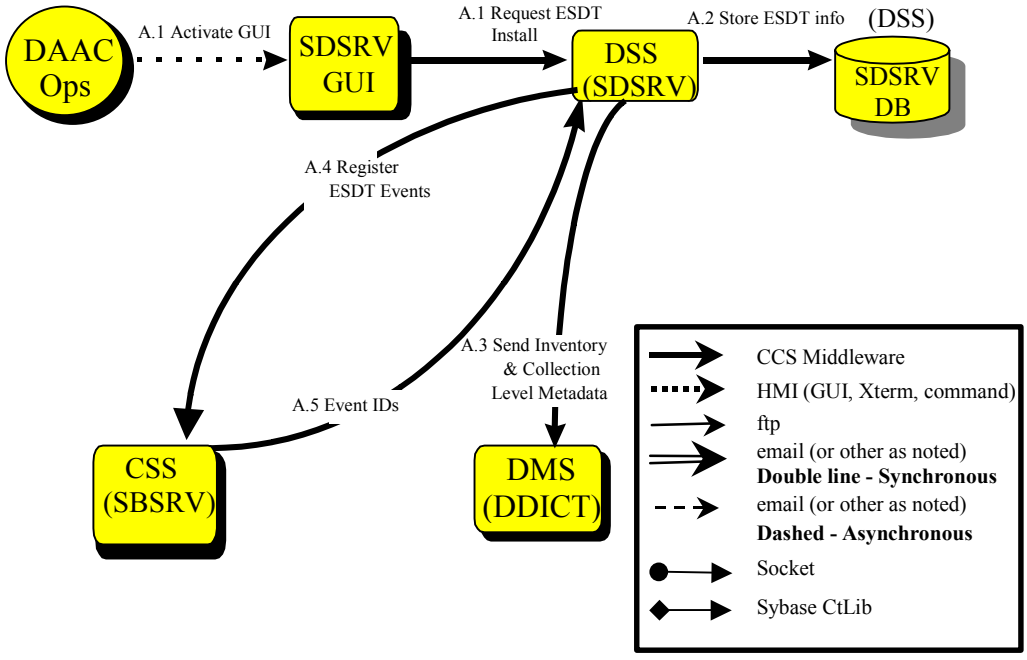

Figure 3.3.4.1-1. Install ESDT Interaction Diagram

3.3.4.2 Install ESDT Interaction Table - Domain View

Table 3.3.4.2-1 provides the Interaction - Domain View: ESDT Installation.

Table 3.3.4.2-1. Interaction Table - Domain View: ESDT Installation

Step	Event	Interface Client	Interface Provider	Data Issues	Step Preconditions	Description
A.1	Activate GUI /Request ESDT Install	DAAC Operator	DSS (SDSRV GUI)	None	DAAC Operator gets the descriptor file location and DLL from the directories specified in the configuration file for the ESDT.	The DAAC Operator brings up the Science Data Server (SDSRV) GUI to install the ESDT. The DAAC Operator, via the Science Data Server GUI, submits a request to the Science Data Server.
A.2	Store ESDT info	DSS (SDSRV)	DSS (SDSRV DB)	None	None	Collection level metadata and configuration information are stored in the Science Data Server's database. Also, DLLs are associated with the ESDT.
A.3	Send Inventory and Collection Level Metadata	DSS (SDSRV)	DMS (DDICT)	None	None	Science Data Server sends collection level and inventory level metadata to the Data Dictionary (DDICT) Server.
A.4	Register ESDT events	DSS (SDSRV)	CSS (SBSRV)	None	None	Science Data Server registers the ESDT events with the Subscription Server (SBSRV).
A.5	Event IDs	CSS (SBSRV)	DSS (SDSRV)	None	None	The Subscription Server sends an event identification to the Science Data Server.

3.3.4.3 Install ESDT Component Interaction Table

Table 3.3.4.3-1 provides the Component Interaction: ESDT Installation.

Table 3.3.4.3-1. Component Interaction Table: ESDT Installation (1 of 3)

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
A.1.1	Startup Science Data Server GUI	DAAC Operator	EcDsSdSrvGui	Xterm	The DAAC Operator invokes the Science Data Server GUI.
A.1.2	Select Add Data Types	DAAC Operator	EcDsSdSrvGui	Xterm	The DAAC Operator selects the Data Types tab and clicks the Add button.

Table 3.3.4.3-1. Component Interaction Table: ESDT Installation (2 of 3)

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
A.1.3	Input ESDT Information	DAAC Operator	EcDsSdSrvGui	Xterm	The DAAC Operator fills in Descriptor Filename.
A.1.4	Submit Add ESDT	EcDsSdSrvGui	EcDsScienceDataServer	CCS Middleware	The DAAC Operator hits the OK button to submit the request to the Science Data Server. The correct Science Data Server is determined via a Server UR, declared in the GUI configuration file.
A.2.1	Descriptor Validation	EcDsScienceDataServer	EcDsScienceDataServer	Internal	The Science Data Server validates the descriptor.
A.2.2	Descriptor and DLL Installation	EcDsScienceDataServer	EcDsScienceDataServer	Internal	The Science Data Server installs the descriptor and Dynamic Link Library (DLL) in the directories specified in its configuration file.
A.2.3	Store ESDT configuration information	EcDsScienceDataServer	Sybase ASE/SQS	CtLib	The Configuration information about the Earth Science Data Type (ESDT) is stored in the Science Data Server's database.
A.2.4	Store ESDT Collection Level Metadata	EcDsScienceDataServer	Sybase ASE/SQS	CtLib	The Collection Level Metadata is stored in the Science Data Server's database.
A.3.1	The Data Dictionary Server receives Collection and Inventory Metadata	EcDsScienceDataServer	EcDmDictServer	CCS Middleware	The Science Data Server sends Collection and Inventory Metadata to the Data Dictionary Server.
A.3.2	Store Collection and Inventory Metadata	EcDmDictServer	Sybase ASE	CtLib	The Data Dictionary server stores the collection and inventory metadata in its database.
A.4.1	Register Events with Subscription Server	EcDsScienceDataServer	EcSbSubServer	CCS Middleware	The Science Data Server registers the ESDT's events with the Subscription Server as ESDT qualifiers.

Table 3.3.4.3-1. Component Interaction Table: ESDT Installation (3 of 3)

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
A.4.2	Store events in Subscription Server	EcSbSub Server	Sybase ASE	CtLib	The Subscription Server stores the ESDT's events in its database.
A.5.1	Send event IDs to Science Data Server	EcSbSub Server	EcDsScience DataServer	CCS Middleware	The Subscription Server sends the Event IDs to the Science Data Server.
A.5.2	Store event IDs in Science Data Server	EcDsScie nceDataS erver	Sybase ASE	CtLib	The Science Data Server stores the Event IDs in its database.

3.3.5 Update ESDT Thread

This thread shows how Earth Science Data Types (ESDTs) are updated in the ECS. The purpose is to have updated ESDT data available in various applications for utilization with advertising, archiving, and subscribing to designated events updated. The update of the ESDT requires an updated descriptor file, an updated Dynamic Link Library file (DLL), if needed, and updated valids, if needed. The updated ESDT descriptor file could contain updated Collection level metadata, Inventory level metadata, updated data services information, or updated data event information. The updated Dynamic Link Library file contains the new services available for the ESDT. The updated valids contain new rules for the attributes.

To accomplish this, the DAAC operator loads updated valids and identifies the updated ESDT. The DSS Science Data Server (SDSRV) sends applicable parts of the updated ESDT to the Data Dictionary (DDICT) Server and the Subscription Server (SBSRV). The Science Data Server also stores the updated ESDT information in its own database and copies the updated ESDT descriptor files and DLL to the target directory.

The ESDTs include data for specific instruments on each mission, external ancillary data, and System data, which includes FDD (orbit and attitude) data.

Update ESDT Thread Preconditions

- The updated ESDT is approved for installation.
- The updated valids are pre-loaded.
- The DAAC Operator knows where the updated descriptor and the updated Dynamic Link Library (DLL) for the ESDT are located.
- Any file space needed for the ESDT or handling the ESDT is not provided explicitly in this scenario. File space is handled as needed by the data servers working with the ESDTs.
- The Science Data Server is placed in the maintenance mode.

3.3.5.1 Update ESDT Interaction Diagram - Domain View

Figure 3.3.5.1-1 depicts the Update ESDT Interaction - Domain View

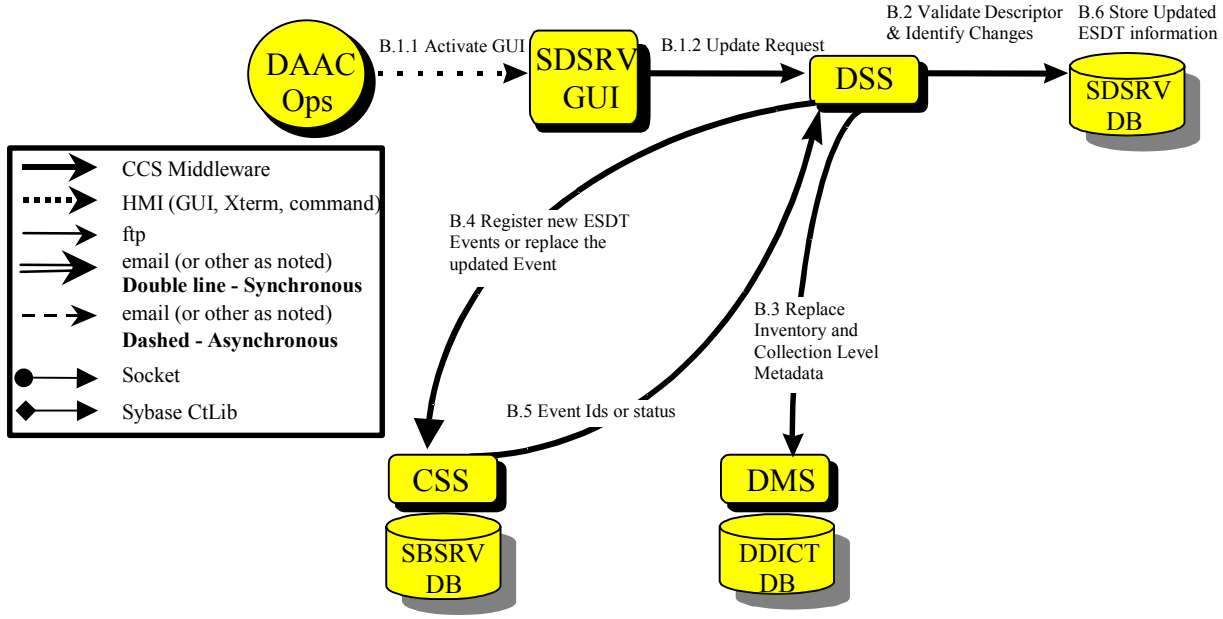

Figure 3.3.5.1-1. Update ESDT Interaction Diagram

3.3.5.2 Update ESDT Interaction Table - Domain View

Table 3.3.5.2-1 provides the Interaction - Domain View: ESDT Update.

Table 3.3.5.2-1. Interaction Table - Domain View: ESDT Update (1 of 2)

Step	Event	Interface Client	Interface Provider	Data Issues	Step Preconditions	Description
B.1	Activate GUI / Update Request	DAAC Operator	DSS (SDSRV)	None	DAAC Operator gets the descriptor file location and DLL from the directories specified in the configuration file for the ESDT.	The DAAC Operator brings up the Science Data Server GUI to update the ESDT. The DAAC Operator, via the Science Data Server GUI, submits a request to the Science Data Server.
B.2	Validate Descriptor and Identify Changes	DSS (SDSRV)	DSS (SDSRV DB)	None	None	The Science Data Server (SDSRV) validates the descriptor and identifies changes.

Table 3.3.5.2-1. Interaction Table - Domain View: ESDT Update (2 of 2)

Step	Event	Interface Client	Interface Provider	Data Issues	Step Preconditions	Description
B.3	Replace Inventory and Collection Level Metadata	DSS (SDSRV) (DSS)	DMS (DDICT)	None	None	The Science Data Server (SDSRV) sends a command to the Data Dictionary (DDICT) Server to replace collection level and inventory level metadata.
B.4	Register new or replace updated ESDT events	DSS (SDSRV)	CSS (SBSRV)	None	None	The Science Data Server registers the new events or sends a command to replace updated Earth Science Data Type (ESDT) events with the Subscription Server (SBSRV).
B.5	Event Ids Or Status	CSS (SBSRV)	DSS (SDSRV)	None	None	The Subscription Server sends Event Identification for new events to the Science Data Server.
B.6	Store Updated ESDT info	DSS (SDSRV)	DSS (SDSRV DB)	None	None	Updated Collection level metadata and new Product Specific Attributes (PSAs) are stored in the Science Data Server's database. Also, Dynamic Link Libraries (DLLs) and descriptor files are copied to the target directories.

3.3.5.3 Update ESDT Component Interaction Table

Table 3.3.5.3-1 provides the Component Interaction: ESDT Update.

Table 3.3.5.3-1. Component Interaction Table: ESDT Update (1 of 2)

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
B.1.1.1	Startup Science Data Server GUI	DAAC Operator	EcDsSdSrvGui	Xterm	The DAAC Operator invokes the Science Data Server GUI.
B.1.1.2	Select Update Data Types	DAAC Operator	EcDsSdSrvGui	Xterm	The DAAC Operator selects the Data Types tab and clicks the Update button.
B.1.1.3	Input ESDT Information	DAAC Operator	EcDsSdSrvGui	Xterm	The DAAC Operator fills in the Descriptor Filename.
B.1.2	Submit Update ESDT	EcDsSdSrvGui	EcDsScienceDataServer	CCS Middleware	The DAAC Operator hits the OK button to submit the request to the Science Data Server. The correct Science Data Server is determined via a Server UR, declared in the GUI configuration file.
B.2.1	Descriptor Validation and Identify Changes	EcDsScienceDataServer	EcDsScienceDataServer	Internal	The Science Data Server validates the descriptor and identifies changes.
B.3.1	The Data Dictionary Server receives Collection and Inventory Metadata information	EcDsScienceDataServer	EcDmDictServer	CCS Middleware	The Science Data Server sends a command to the Data Dictionary Server to replace updated Collection and Inventory Metadata information.
B.3.2	Updated Collection and Inventory Metadata replaced in the Data Dictionary Server	EcDmDictServer	Sybase ASE	CtLib	The Data Dictionary server replaces the collection and inventory metadata in its database.

Table 3.3.5.3-1. Component Interaction Table: ESDT Update (2 of 2)

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
B.4.1	Register new events or replace updated vents with Subscription Server	EcDsScienceDataServer	EcSbSubServer	CCS Middleware	The Science Data Server registers the ESDT's new events including ESDT qualifiers or replaces updated events with the Subscription Server.
B.4.2	Store events in Subscription Server	EcSbSubServer	Sybase ASE	CtLib	The Subscription Server stores the ESDT's events in its database.
B.5.1	Send event IDs to Science Data Server	EcSbSubServer	EcDsScienceDataServer	CCS Middleware	The Subscription Server sends the Event IDs to the Science Data Server.
B.5.2	Store event Ids in Science Data Server	EcDsScienceDataServer	Sybase ASE	CtLib	The Science Data Server stores the Event IDS in its database.
B.6.1	Descriptor and DLL Installation	EcDsScienceDataServer	EcDsScienceDataServer	Internal	The Science Data Server installs the descriptor and DLL in the target directories specified in its configuration file.
B.6.2	Store ESDT Collection Level Metadata	EcDsScienceDataServer	Sybase ASE/SQS	CtLib	The Collection Level Metadata is stored in the Science Data Server's database.

3.3.6 (Deleted)

3.3.7 (Deleted)

3.3.8 Import ESDTs Thread

This Thread shows how Earth Science Data Types (ESDTs) are imported into the ECS system from other protocols, namely ASTER GDS. The purpose is to have non-ECS ESDT metadata available to ECS clients so they may have access to services relating to those ESDTs. The import of an ESDT requires the existence of a valids file from an external protocol. The valids file contains Collection level and Inventory level metadata and data services information, written in ODL obeying a format specified by the external protocol.

To accomplish this, the DAAC operator supplies the Maintenance Tool with a valids file for import. The Data Dictionary (DDICT) Server is then supplied with metadata pertaining to that ESDT.

Import ESDT Preconditions

- The valids file has been acquired from the external protocol.

3.3.8.1 Import ESDT Interaction Diagram - Domain View

Figure 3.3.8.1-1 depicts the Import ESDT Interaction - Domain View

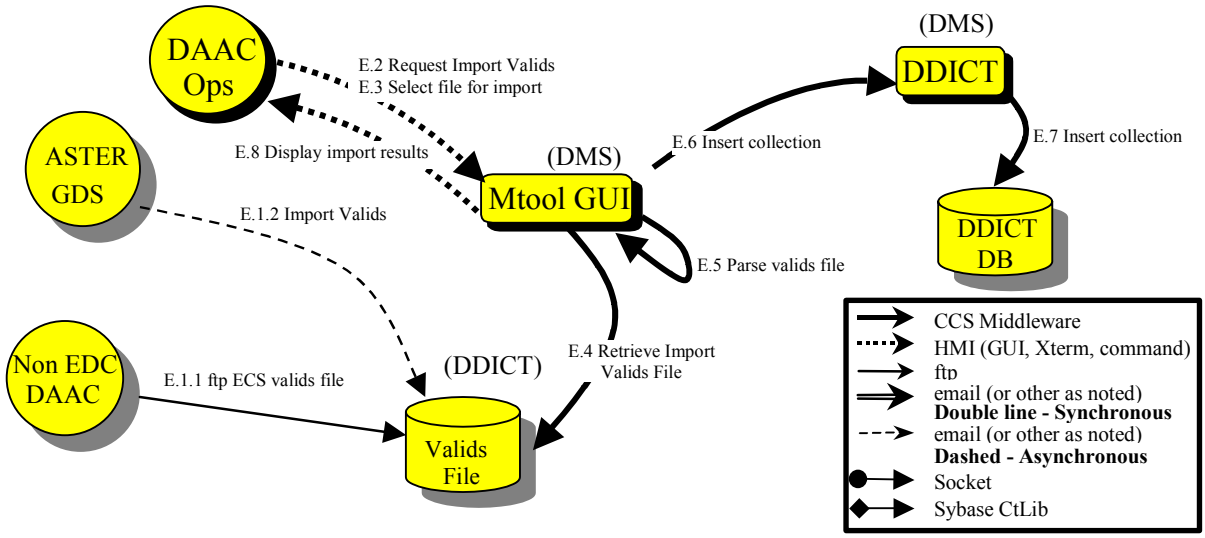

Figure 3.3.8.1-1. Import ESDT Interaction Diagram

3.3.8.2 Import ESDT Interaction Table - Domain View

Table 3.3.8.2-1 provides the Interaction - Domain View: ESDT Import.

Table 3.3.8.2-1. Interaction Table - Domain View: ESDT Import

Step	Event	Interface Client	Interface Provider	Data Issues	Step Preconditions	Description
E.1	ftp ECS valids file / Import Valids	Non EDC DAAC / ASTER GDS	DMS (DDICT Valids File)	None	Remote Operator has transferred a valids file.	A Remote DAAC operator transfers a valids file.
E.2	Request Import Valids	DAAC Operator	DMS (Mtool GUI)	None	DAAC Operator knows how to run the Data Dictionary Maintenance Tool.	The DAAC Operator brings up the Data Dictionary (DDICT) Maintenance Tool (Mtool) and selects the import valids tab.
E.3	Select file for import	DAAC operator	DMS (Mtool GUI)	None	DAAC Operator has a valids file to import.	The DAAC operator selects appropriate filename for import.
E.4	Retrieve Import Valids file	DMS (Mtool GUI)	DMS (DDICT Valids File)	None	None	Access and read selected file.
E.5	Parse valids file	DMS (Mtool GUI)	DMS (Mtool GUI)	None	File is accessible.	Read in and create internal representation of the valids file.
E.6	Insert collection	DMS (Mtool GUI)	DMS (DDICT)	None	Valids file is correct.	Send insert collection to the Data Dictionary Server.
E.7	Insert collection	DMS (DDICT)	DMS (DDICT DB)	None	None	Insert the valids into the Data Dictionary Server database.
E.8	Display import results	DMS (Mtool GUI)	DAAC operator	None	None	Show DAAC Operator the results of the import.

3.3.8.3 Import ESDT Component Interaction Table

Table 3.3.8.3-1 provides the Component Interaction: ESDT Import.

Table 3.3.8.3-1. Component Interaction Table: ESDT Import

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
E.1.1	Receive a valids file from a Non-EDC DAAC	Non EDC DAAC Operator	Valids File	Ftp	A Non-EDC DAAC operator transfers a valids file to EDC via Ftp.
E.1.2	Receive a valids file from ASTER GDS	ASTER GDS	Valids File	E-mail	A valids file is transferred to EDC from ASTER GDS via e-mail.
E.2.1	Startup Data Dictionary Maintenance Tool GUI	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator invokes the Data Dictionary Maintenance Tool GUI.
E.2.2	Select 'Import Valids File' tab	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator selects the 'Import Valids File' tab.
E.3.1	Supply import filename	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator types in the filename.
E.4.1	Retrieve Import Valids File	EcDmDd MaintenanceTool	Valids File (Sybase ASE)	Xterm	Access and read selected file.
E.5.1	Click on check button	DAAC operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator clicks on Check button; the tool then checks the file contents for syntactic and semantic errors.
E.6.1	Send ESDT metadata to the Data Dictionary Server	EcDmDd MaintenanceTool	EcDmDictServer	CCS Middleware	Schema update sent to the Data Dictionary server relating to ESDT. This generates a new collection entity within the schema.
E.7.1	Send ESDT services to ADSRV	EcDmDict Server	DDICT DB (SQS Sybase ASE)	CCS Middleware	Update database.
E.8.1	View results	EcDmDd MaintenanceTool	DAAC Operator	Xterm	Display the results of the import to the DAAC operator.

3.3.9 Export ESDT Thread

This scenario shows how Earth Science Data Types (ESDTs) are exported in the ECS system to other protocols, namely V0-IMS and ASTER GDS. The purpose is to have ECS ESDT metadata available to other protocols so that their clients may have access to ECS services relating to those ESDTs. The export of an ESDT requires the successful install of that ESDT, the creation of a

valids file and the sending of that file to the external protocol. The valids file contains Collection level and Inventory level metadata and data services information, written in ODL obeying a format specified by the destination protocol.

To accomplish this, the DAAC operator identifies the ESDT(s) to be exported and instructs the Maintenance Tool to generate a valids file for them based on the protocol determined by the operator. The Data Dictionary (DDICT) Server is queried for metadata. The result of this query is then placed within operator-supplied valids file in a format compliant with the specified protocol. This file is then sent via ftp to the external protocol.

Export ESDT Thread Preconditions

- The ESDT has been installed by the Science Data Server
- The DAAC Operator knows which ESDTs to export

3.3.9.1 Export ESDT Interaction Diagram - Domain View

Figure 3.3.9.1-1 depicts the Export ESDT Interaction - Domain View

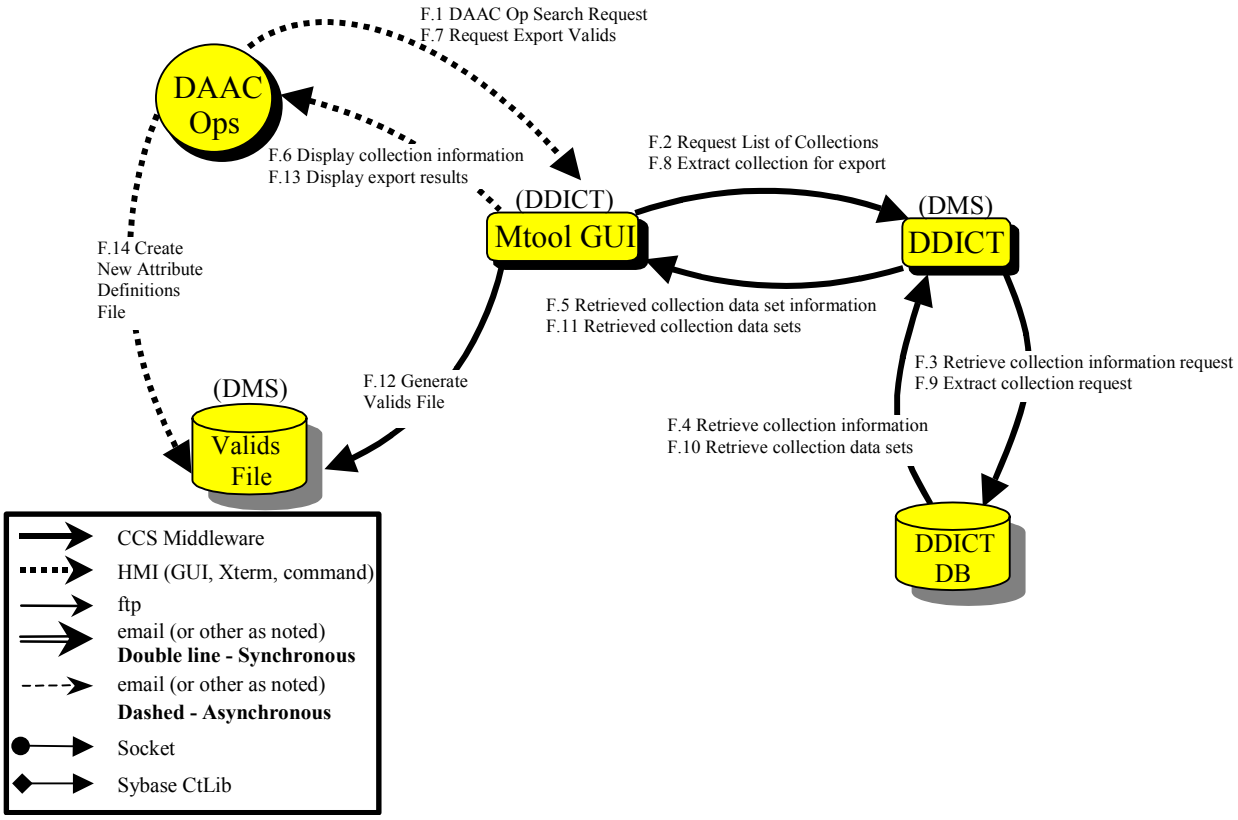

Figure 3.3.9.1-1. Export ESDT Interaction Diagram

3.3.9.2 Export ESDT Interaction Table - Domain View

Table 3.3.9.2-1 provides the Interaction - Domain View: ESDT Export.

Table 3.3.9.2-1. Interaction Table - Domain View: ESDT Export (1 of 2)

Step	Event	Interface Client	Interface Provider	Data Issues	Step Preconditions	Description
F.1	DAAC Op Search request	DAAC Operator	DMS (Mtool GUI)	None	DAAC Operator knows how to run the Data Dictionary Maintenance Tool.	The DAAC Operator brings up the Data Dictionary Maintenance Tool and selects the export valids tab. The DAAC Operator requests the list of available collections.
F.2	Request list of collections	DMS (Mtool GUI)	DMS (DDICT)	None	Data Dictionary Maintenance Tool is up.	The Maintenance tool sends a query to the Data Dictionary Server for metadata pertaining to each ESDT.
F.3	Retrieve collection information request	DMS (DDICT)	DMS (DDICT DB)	None	None	Request list of collections.
F.4	Retrieve collection information	DMS (DDICT DB)	DMS (DDICT)	None	None	Retrieve collection information.
F.5	Retrieved data set collection information	DMS (DDICT)	DMS (Mtool GUI)	None	None	Retrieve data set collection information.
F.6	Display collection information	DMS (Mtool GUI)	DAAC Operator	None	None	Display collection information.
F.7	Request Export Valids	DAAC Operator	DMS (Mtool GUI)	None	None	Select collection(s) for export.
F.8	Extract collection for export	DMS (Mtool GUI)	DMS (DDICT)	None	None	Request collection(s) for export.
F.9	Extract collection request	DMS (DDICT)	DMS (DDICT DB)	None	None	Retrieve collection(s).
F.10	Retrieve collection data sets	DMS (DDICT DB)	DMS (DDICT)	None	None	Retrieve collection(s).

Table 3.3.9.2-1. Interaction Table - Domain View: ESDT Export (2 of 2)

Step	Event	Interface Client	Interface Provider	Data Issues	Step Preconditions	Description
F.11	Retrieved collection data sets	DMS (DDICT)	DMS (Mtool GUI)	None	None	Retrieve collection(s).
F.12	Generate Valids File	DMS (Mtool GUI)	DMS (Valids File)	None	None	Construct valids structure for export.
F.13	Display Export results	DMS (Mtool GUI)	DAAC Operator	None	None	The DAAC Operator can view the export results.
F.14	Create new attribute definitions file	DAAC Operator	DMS (Valids File)	None	None	The DAAC Operator selects the filename for the results.

3.3.9.3 Export ESDT Component Interaction Table

Table 3.3.9.3-1 provides the Component Interaction: ESDT Export.

Table 3.3.9.3-1. Component Interaction Table: ESDT Export (1 of 3)

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
F.1.1	Startup Data Dictionary Maintenance Tool GUI	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator invokes the Data Dictionary Maintenance Tool GUI.
F.1.2	Select 'Export Valids File' tab	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator selects the 'Export Valids File' tab.
F.1.3	Select export protocol	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator uses pick list to determine the export protocol.
F.1.4	Click on ESDT 'selection criteria' button	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator clicks on button.
F.1.5	Select ESDT search constraints	DAAC operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator selects search constraints.
F.2.1	ESDT metadata requested from the Data Dictionary Server	EcDmDdMaintenanceTool	EcDmDictServer	CCS Middleware	The Maintenance tool sends a query to the Data Dictionary server for metadata pertaining to each ESDT.

Table 3.3.9.3-1. Component Interaction Table: ESDT Export (2 of 3)

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
F.3.1	The Data Dictionary Server queries database	EcDmDictServer	DDICT DB	CCS Middleware	The Data Dictionary Server queries the database for metadata.
F.4.1	The Data Dictionary Server receives metadata	DDICT DB	EcDmDictServer	CCS Middleware	The Data Dictionary Server retrieves the results of the query.
F.5.1	Return ESDT metadata	EcDmDictServer	EcDmDdMaintenanceTool	CCS Middleware	The Data Dictionary Server sends the results of the query to the Maintenance tool.
F.6.1	Display the list of collections	EcDmDdMaintenanceTool	DAAC Operator	Xterm	The DAAC Operator can view the results of the query.
F.7.1	Select ESDTs to export	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator selects the ESDTs he/she wishes to export.
F.8.1	ESDT metadata requested from the Data Dictionary Server	EcDmDdMaintenanceTool	EcDmDictServer	CCS Middleware	The Maintenance tool sends a query to the Data Dictionary Server for metadata pertaining to each ESDT.
F.9.1	The Data Dictionary Server queries database	EcDmDictServer	DDICT DB	CCS Middleware	The Data Dictionary Server queries the database for metadata.
F.10.1	The Data Dictionary Server receives metadata	DDICT DB	EcDmDictServer	CCS Middleware	The Data Dictionary Server retrieves the results of the query.
F.11.1	Return ESDT metadata	EcDmDictServer	EcDmDdMaintenanceTool	CCS Middleware	The Data Dictionary Server sends the results of the query to the Maintenance tool.
F.12.1	Select file for export target	EcDmDdMaintenanceTool	EcDmDdMaintenanceTool Valids File (SQS Sybase ASE)	CCS Middleware	The DAAC Operator provides the Data Dictionary Maintenance tool with the filename of the export file.

Table 3.3.9.3-1. Component Interaction Table: ESDT Export (3 of 3)

Step	Event	Interface Client	Interface Provider	Interface Mech.	Description
F.12.2	Form File	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The maintenance tool forms a file containing the ESDT metadata and servers in a protocol determined by the DAAC operator.
F.13.1	Display the export results	EcDmDdMaintenanceTool	EcDmDdMaintenanceTool DAAC Operator	Xterm	The DAAC Operator can view the results of the export request.
F.14.1	Save Export file	DAAC Operator	EcDmDdMaintenanceTool	Xterm	The DAAC Operator clicks a button to save the export file.

3.4 System Start-up/Shutdown

The procedures for start up and shutdown are described in the Mission Operations Procedures document, 611-CD-610, Section 3.2, System Startup and Shutdown. In the ECS Project Training Material document, 625-CD-604, Volume 4: System Administration in the section titled System Startup and Shutdown, the process for starting and shutting down the system is also described.